Project Report on

# 9 Men Morris Game

**Team Name/no**

Four Sure -29

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

**Submitted by-**

Shubham Kathiriya (Roll no : 2023201050)

Sneha Arora (Roll no: 2023201025)

Soham Kale (Roll no: 2023202018)

Soham Ghosh (Roll no: 2023202011)

**Instructor**

Soumitra Ghosh

**Course**

Mtech. CSE/CSIS

**Academic Year**

(2023 -2024)

## Abstract

*Nine Men's Morris, also known as "Mill," is an ancient strategy board game that dates back to antiquity. The game is played on a grid-based board comprising three*

*concentric squares with lines intersecting at key points, forming a total of 24 intersections. Two players each control nine pieces and take turns placing them on the board and subsequently moving them to form lines of three, called "mills." Creating a mill allows a player to capture one of the opponent's pieces. The primary objectives are to strategically position pieces, block opponents' moves, and create or break mills strategically. The game progresses through the phases of placement and movement until one player is reduced to two pieces or is unable to make a legal move, resulting in victory for the other player. Nine Men's Morris offers a balance of tactical depth and simplicity, making it a timeless classic that has persisted across cultures and centuries. Its enduring appeal lies in its accessibility, yet it challenges players to devise intricate strategies to outmaneuver their opponents.*

# Table of Contents

# User Interface(Frontend)

## 1.1 Game – Board

**create_board** function initializes and returns an array of cell objects representing each cell on the game board. Each cell is created with a unique number, x-coordinate, y-coordinate, and initializations for neighbors, vertical cells, and horizontal cells.

The **cell class** represents an individual cell on the game board. Each cell has:

- cell_number: A unique identifier for the cell.
- x_cordinate and y_cordinate: Coordinates specifying the position of the cell.
- neighbours, vertical_cell, and horizontal_cell: Arrays to store neighboring cells.
- isEmpty: A boolean flag indicating whether the cell is occupied.
- player_number: The player number (null if the cell is empty).
- player_colour: The color of the player's piece.

The class includes methods like add_neighbours, add_vertical_cell, and add_horizontal_cell for adding connections between cells, and display for rendering the cell on the canvas.The game board is created with predefined cell positions and connections.

**create_game_board** function utilizes the p5.js library to draw the initial game board layout. It draws squares and lines based on the game board's structure and then iterates through each cell to display them using the display method.

**preload and setup**, functions are part of the p5.js setup and are responsible for loading the background image and initializing the canvas. The setup function also checks for a saved board state in local storage and either loads it or creates a new board accordingly.Images like "board.jpg" are preloaded for background usage.The code uses the p5.js library for rendering and drawing graphics on the canvas.

## 1.2 Timer

timer container with three sections: left timer, controls, and right timer. The left and right timers include buttons labeled "Player 1" and "Player 2," respectively, with corresponding time displays. The controls section features a "Morris Clock" heading and a form group containing an input field for setting the timer duration in minutes, along with "Pause" and "Reset" buttons to control the timer.

The timers are designed to work alternately based on player turns, synchronizing with the game's progress, associated with moving game pieces.making it a turn-based game, where players take turns making moves, and the timer adapts accordingly.

## 1.3 Player Status

The UI is structured using flex containers and consists of two main sections, each representing a player's status.The section has a light gray background (bg-gray-200) and padding (p-4). And is Identical for Player-1 and Player-2 but updating their respective properties.

Player Information: Inside the section, there is a white rounded container with a shadow, containing:

- Player Name: Displayed in a large font (text-5xl), including the text "Player" and a badge with a blue background and the label "One" (bg-blue-100).
- Remaining Pieces: A container (player_one_remaining_piece) to display information about the remaining game pieces for Player One.
- Out Pieces: A container (player_out_one_piece) to display information about the pieces that Player One has already placed on the board

## 1.4 Save, Load, Reset, How to play

1. Save Button (#save):
    - Appearance: Styled as a large primary button with a height of 70 pixels and a font size of 29 pixels.
    - Functionality: Intended for saving the game state or pausing the game. When clicked, it triggers a function to save the current state of the game, allowing players to resume from the same point later.
2. Load Button (#Load):
    - Appearance: Styled as a large primary button with a height of 70 pixels and a font size of 29 pixels.
    - Functionality: Intended for loading a previously saved game state. Clicking this button triggers a function to retrieve and load the saved state, allowing players to continue the game from where they left off.
3. Reset Button (#gameReset):
    - Appearance: Styled as a large danger-themed button with a height of 70 pixels and a font size of 29 pixels.
    - Functionality: Intended for resetting the game board. Clicking this button triggers a function to reset the entire game board to its initial state, removing any progress or changes made during the current session.
4. How to Play (#howToPlay) :
    - Appearance: Styled as a large primary button with a height of 70 pixels and a font size of 29 pixels.
    - Functionality: Intended for resetting the game board. Clicking this button triggers a function to show all the rules to the game

## 1.5 Global Styling

Font: The @import url(https://fonts.googleapis.com/css?family=Roboto); rule imports the Roboto font from Google Fonts and applies it globally to the entire document using font-family: "Roboto", sans-serif;.

Background: The background-image, background-size, and background-color properties are applied to the html and body elements, defining the background appearance for the entire page. In this case, an image ("Untitled.jpeg") is set as the background with a specified size and color.

# Game Logic (backend)

Game logic have been implemnted mainly in chunks where we are handling the implementation of the game .

## 2.1 Core logic

These functions collectively handle the core mechanics of placing a piece, checking neighbors, and updating the game state based on player actions.

### 2.1.1 Mouse Click Event Handling (mouse_click_event):

- The mouse_click_event function is an event listener that responds to mouse clicks on the game board.
- It iterates through all cells to determine the cell clicked based on its proximity to the mouse coordinates.
- Depending on the game phase (placing or sliding), different actions are taken, such as handling mill formation, moving pieces, and triggering sound effects.

### 2.1.2 Moving Phase (moving_phase):

- The moving phase allows players to select their pieces and slide them to adjacent empty cells.
- The first click identifies the player's piece, and the second click determines the target cell for sliding.
- Validations include ensuring that the selected cells are neighbors and checking for empty neighbors during the first click.
- The slidePiece function swaps the content of the current cell and the target cell if the target cell is empty.

### 2.1.3 Mill Formation (checkForMill and handle_mill):

- checkForMill checks for mill formation after a successful slide. It evaluates both horizontal and vertical alignments of three pieces.
- handle_mill processes mill formation, removing an opponent's piece and updating the game state.
- Mills are checked and handled separately for horizontal and vertical alignments

### 2.1.4 handleCellClick(clickedCell)

This function is responsible for handling the logic when a player clicks on a cell in the game board. Here's a breakdown of its key components:

1. Checking if the Cell is Already Colored:
    - It starts by checking if the clicked cell is already occupied by a player's piece. If the cell is not empty, it logs a message and returns, indicating that no further action is needed.
2. Checking if Maximum Pieces Placed:

- It then checks if the current player has already placed the maximum allowed number of pieces (9 in this case). If so, it logs a message and returns, preventing additional pieces from being placed.

3. Placing a Piece:
    - If the cell is empty and the player has not reached the maximum piece limit, it places the player's piece in the clicked cell, updates the player's piece count, and decreases the count of available pieces on hand.
    - It then checks for mill formation after placing the piece and handles it accordingly.
4. Switching Players and Starting Timers:
    - After placing a piece, it switches to the other player's turn and starts the corresponding timer.
    - If both players have placed all their pieces, it sets a flag (full_now) to true.
5. Updating Game Board Display:
    - The function calls create_game_board() to update the display of the game board with the new piece placements.

### 2.1.5 getAdjacentCells(clickedCell)

This utility function takes a cell object (clickedCell) as input and returns an array of neighboring cells. It's used to determine the adjacent cells of a clicked cell.

1. Finding Adjacent Cells:
    - It iterates through the neighbours array of the clickedCell and finds the corresponding cell objects in the GAME_BOARD.all_cell array.
    - It builds an array (adjacentCells) containing these neighboring cell objects.
2. Handling Missing Neighbors:
    - If a neighbor cell is not found, it logs a message indicating that the neighbor cell was not found.
3. Returning the Result:
    - The function returns the array of adjacent cells.

### 2.1.6 printNeighbors(clickedCell)

This function logs the neighbors of a clicked cell. It uses the getAdjacentCells function to obtain the neighboring cells and prints their cell numbers.

1. Logging Neighbors:
    - It calls getAdjacentCells to get the array of neighboring cells.
    - It then logs the cell numbers of the neighbors.
2. Handling Missing Neighbors:

- If a neighbor cell is not found, it logs a message indicating that the neighbor cell was not found.

## 2.2 Winning logic

These functions collectively handle the win conditions based on piece counts, blocking situations, and timer expiration.

### 2.2.1 is_block(player_num)

This function checks if all the neighboring cells of a player's pieces on the board are filled. If they are, it returns true, indicating that the player's pieces are blocked from moving.

1. Iterating Over Cells:
    - It iterates through all the cells on the game board (GAME_BOARD.all_cell).
2. Checking Player's Pieces:
    - For each cell that contains a piece belonging to the specified player (player_num), it checks if all its neighboring cells are filled.
3. Updating all_block:
    - If any neighboring cell is found to be empty (null), it sets all_block to false.
4. Returning Result:
    - It returns the final result (all_block). If any piece of the player is not blocked, it returns false; otherwise, it returns true.

### 2.2.2 win_condition(player_obj)

This function checks two conditions to determine if a player has won:

1. Condition 1 - Out Piece Count:
    - If the number of pieces taken out (out_piece) by the player is equal to TOTAL_PIECE - 2, it logs a message and returns true, indicating a win.
2. Condition 2 - On Board and Blocking Check:
    - If the player has no pieces on the board (on_board == 0), it logs a message and returns false, indicating no win.
    - Otherwise, it calls the is_block function to check if all of the player's pieces are blocked. If they are, it logs a message and returns true, indicating a win.

### 2.2.3 checkGameStatus()

This function is likely used to check the game status based on the timers (t1 and t2). It logs a message declaring the winner if one of the players' time is up.

1. Checking Timer for Player 1:
    - If the timer (t1) for Player 1 reaches 0 seconds, it logs a message declaring that Player 2 wins because Player 1's time is up.

- It removes the click event listener, adds a win sound, and pauses the timers.

2. Checking Timer for Player 2:

   - If the timer (t2) for Player 2 reaches 0 seconds, it logs a message declaring that Player 1 wins because Player 2's time is up.
   - It removes the click event listener, adds a win sound, and pauses the timers.

## 2.3 Save State

These functions collectively provide the user with the ability to save, reset, and load game states, offering a way to continue or restart the game as needed

### 2.3.1 save_game()

Saves the current state of the game, including the game board and the array of cells, to the local storage.

- Uses localStorage.setItem to store the serialized JSON representation of the GAME_BOARD and cellsArray.
- Converts the JavaScript objects (GAME_BOARD and cellsArray) into JSON strings using JSON.stringify.
- Logs a message to the console indicating that the game has been saved.

### 2.3.2 reset_game()

Resets the game to its initial state by reloading the current location.

- Uses location.reload() to reload the current page.

### 2.3.3 load_game()

Loads a previously saved game state from the local storage.

- Uses localStorage.getItem to retrieve the serialized JSON representations of the saved game board (saved_board) and cells array (saved_cells_array).
- Parses the JSON strings back into JavaScript objects using JSON.parse.
- Updates the current game board (GAME_BOARD) and cells array (cellsArray) with the saved values.
- Calls the create_game_board function to recreate and display the game board based on the loaded state.

## 2.4 Message status

This section is used to print the mesages in the game interface

### 2.4.1 win_message(win_player)

Displays a win message based on the winning player.

win_player: An integer representing the winning player (1 or 2).

- Selects the HTML element with the ID "win" using document.querySelector.
- Updates the content of the selected element based on the winning player.
- If win_player is 1, sets the innerHTML to "Player 1 wins :)".
- If win_player is 2, sets the innerHTML to "Player 2 wins :)".

### 2.4.2 turn_message(temp_turn)

Displays a message indicating the current turn player.

temp_turn: An integer representing the current turn player (1 or 2).

- Selects the HTML element with the ID "win" using document.querySelector.
- Updates the content of the selected element based on the current turn player.
- If temp_turn is 1, sets the innerHTML to "Turn: Player 1".
- If temp_turn is 2, sets the innerHTML to "Turn: Player 2".

### 2.4.3 mill_formation_message(temp_turn)

Displays a message indicating that a mill has been formed and instructs to remove a piece from the opponent.

temp_turn: An integer representing the current turn player (1 or 2).

- Selects the HTML element with the ID "win" using document.querySelector.
- Updates the content of the selected element based on the current turn player.
- If temp_turn is 1, sets the innerHTML to "Player 1 has formed a mill, remove one piece from Player 2".
- If temp_turn is 2, sets the innerHTML to "Player 2 has formed a mill, remove one piece from Player 1".

### 2.4.4 default_message()

Displays a default message.

- Selects the HTML element with the ID "win" using document.querySelector.
- Sets the innerHTML of the selected element to "Default"

## Contributions

User Interface**:** Sneha

Placing Place: Shubham

Piece Move: Soham Kale

Mill/Win: Shubham, Soham Kale

Save State: Shubham

Timer: Soham Kale, Sneha, Soham Ghosh

Sound, Interactive: Sneha, Soham Ghosh

Documentation: Sneha, Soham Kale