Owen Lindsey

Professor Hughes, Bill

CST-350 Milestone

11/24/2024

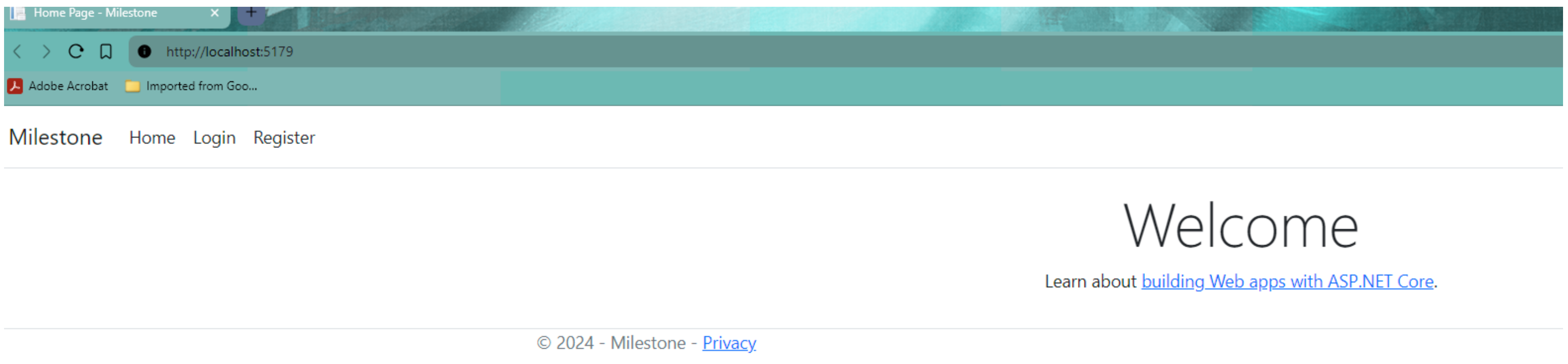
Part 1 Loom video and GitHub links:

Loom recording

Github link

Part 2 Screenshots of application:

The home screen shows the ability to either login or register before the user can enter the site and play the game.

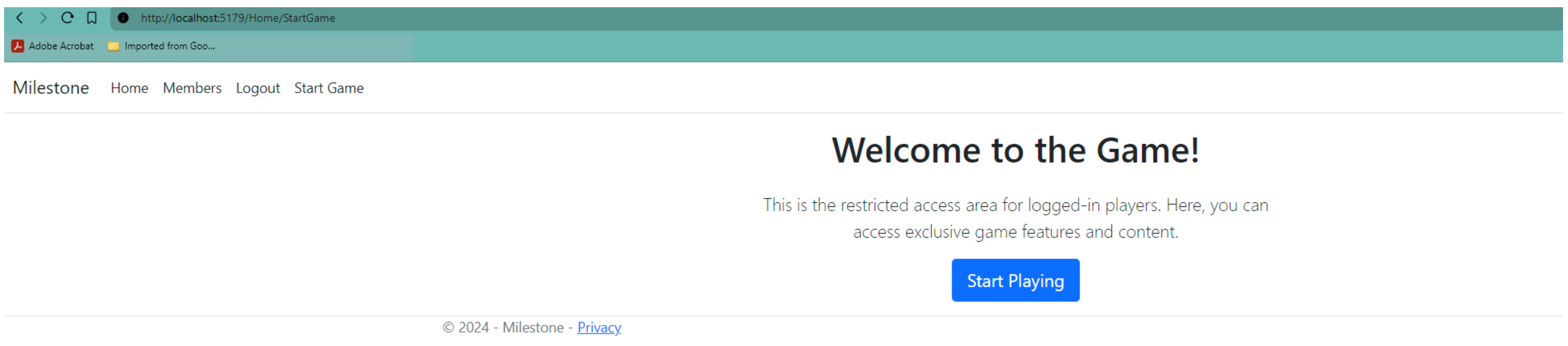


Welcome

Learn about building Web apps with ASP.NET Core.

© 2024 - Milestone - Privacy

Once logged in the user can then start a mine sweeper game by selecting start game in the navigation bar.

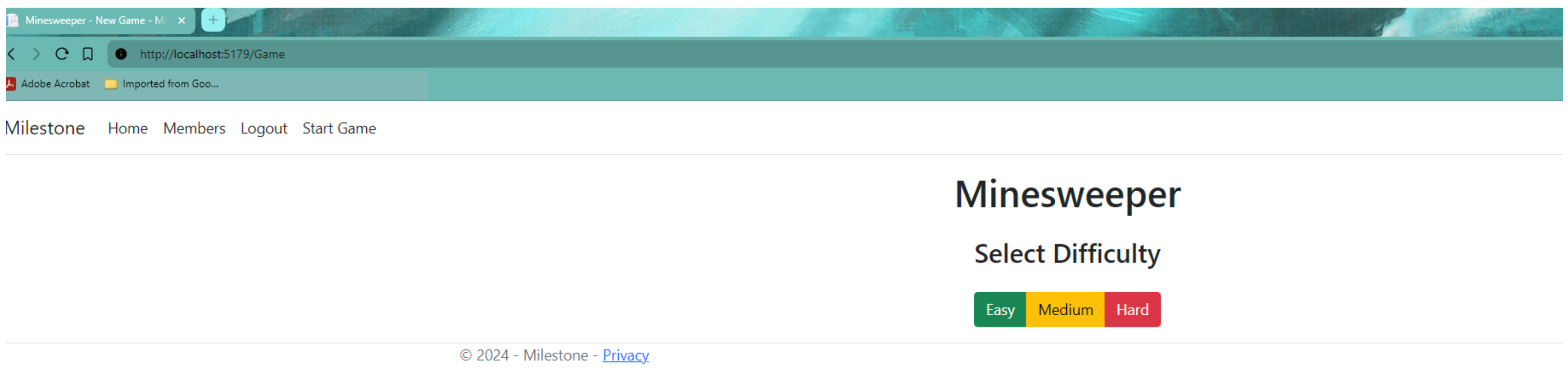


Welcome to the Game!

This is the restricted access area for logged-in players. Here, you can access exclusive game features and content.

© 2024 - Milestone - Privacy

Once the user selects the Start Playing button, they will be greeted with a difficulty selection screen from Game/Index.cshtml view.



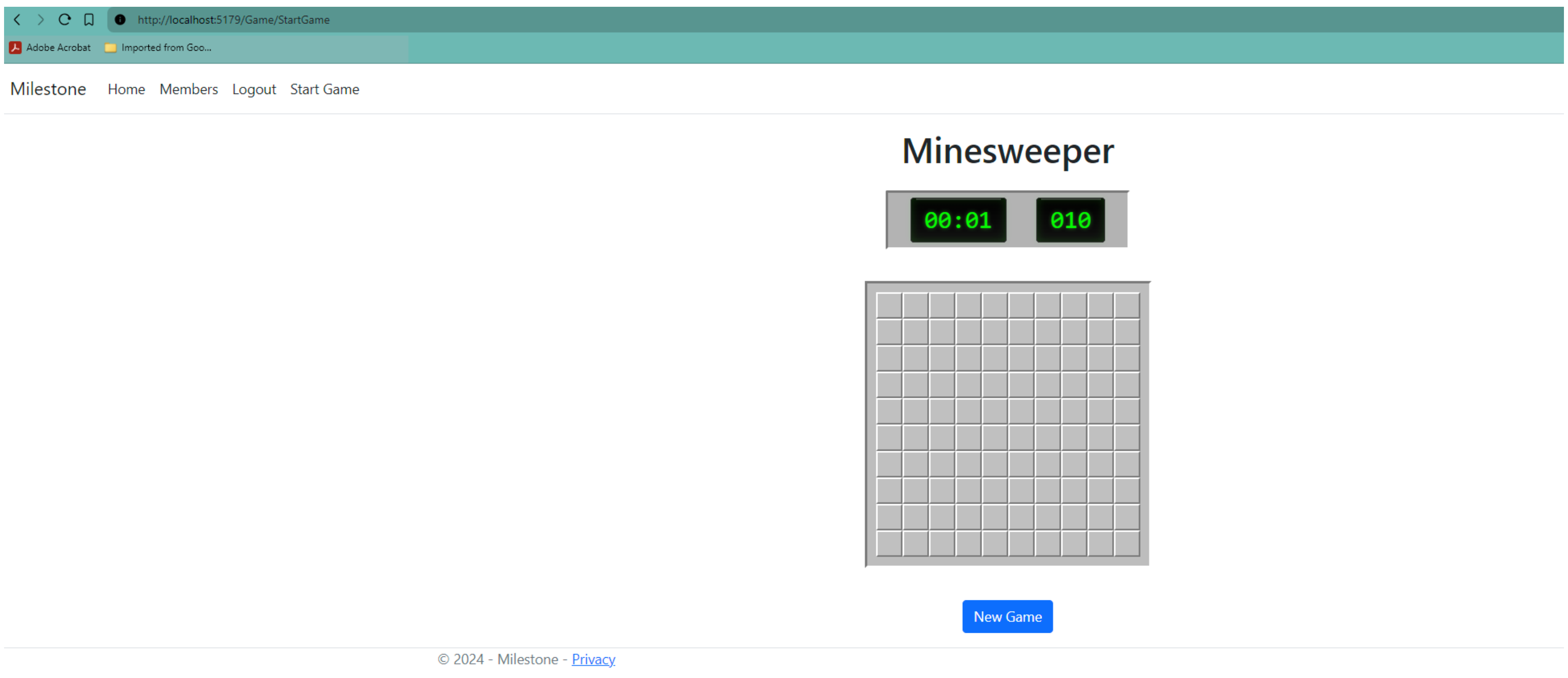
Minesweeper

Select Difficulty

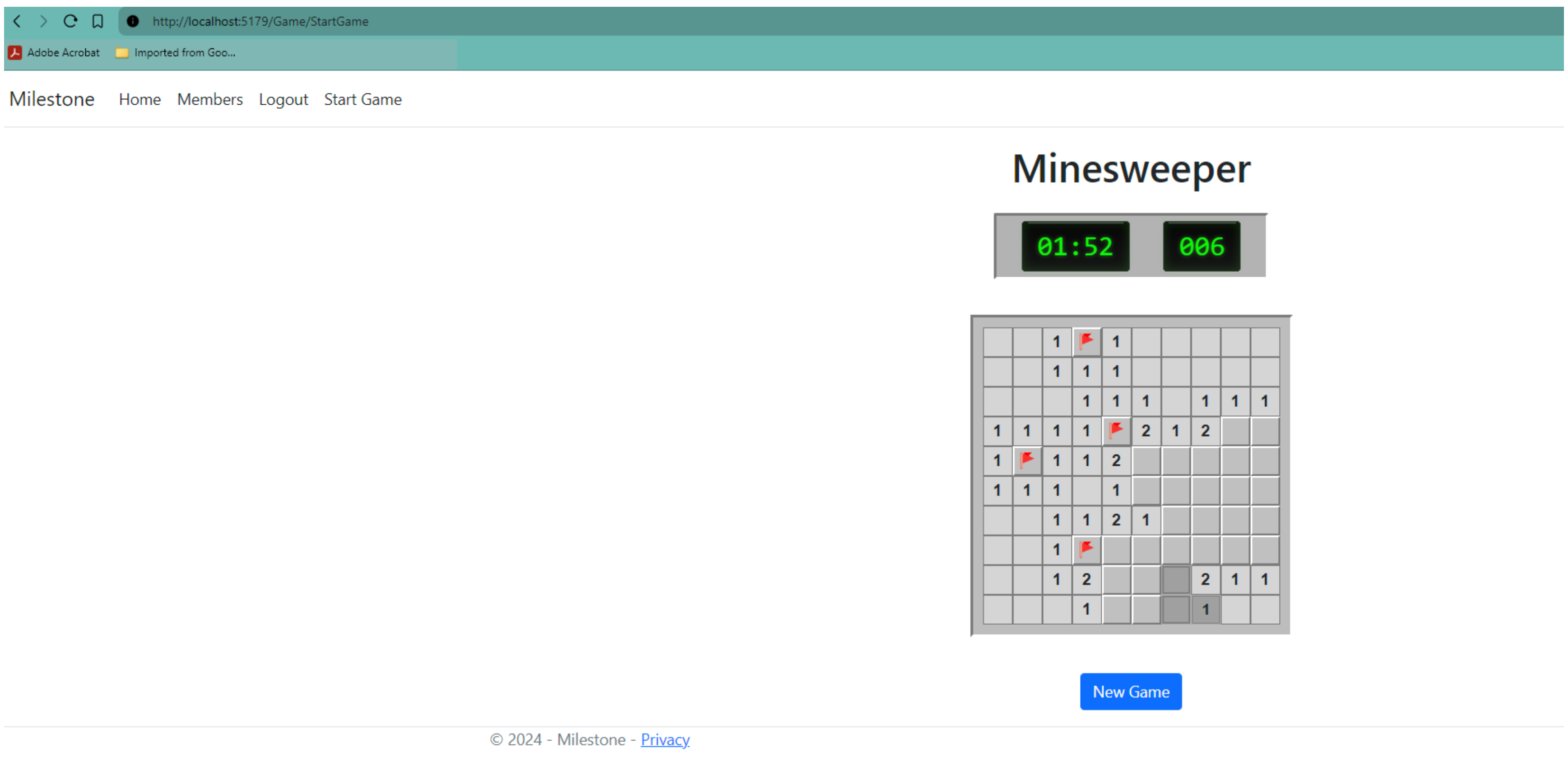
© 2024 - Milestone - Privacy

Part 2 Screenshots of application:

Once the user has selected a difficulty, the minesweeper game will start. The number of mines and the timer will be displayed above the minesweeper board.

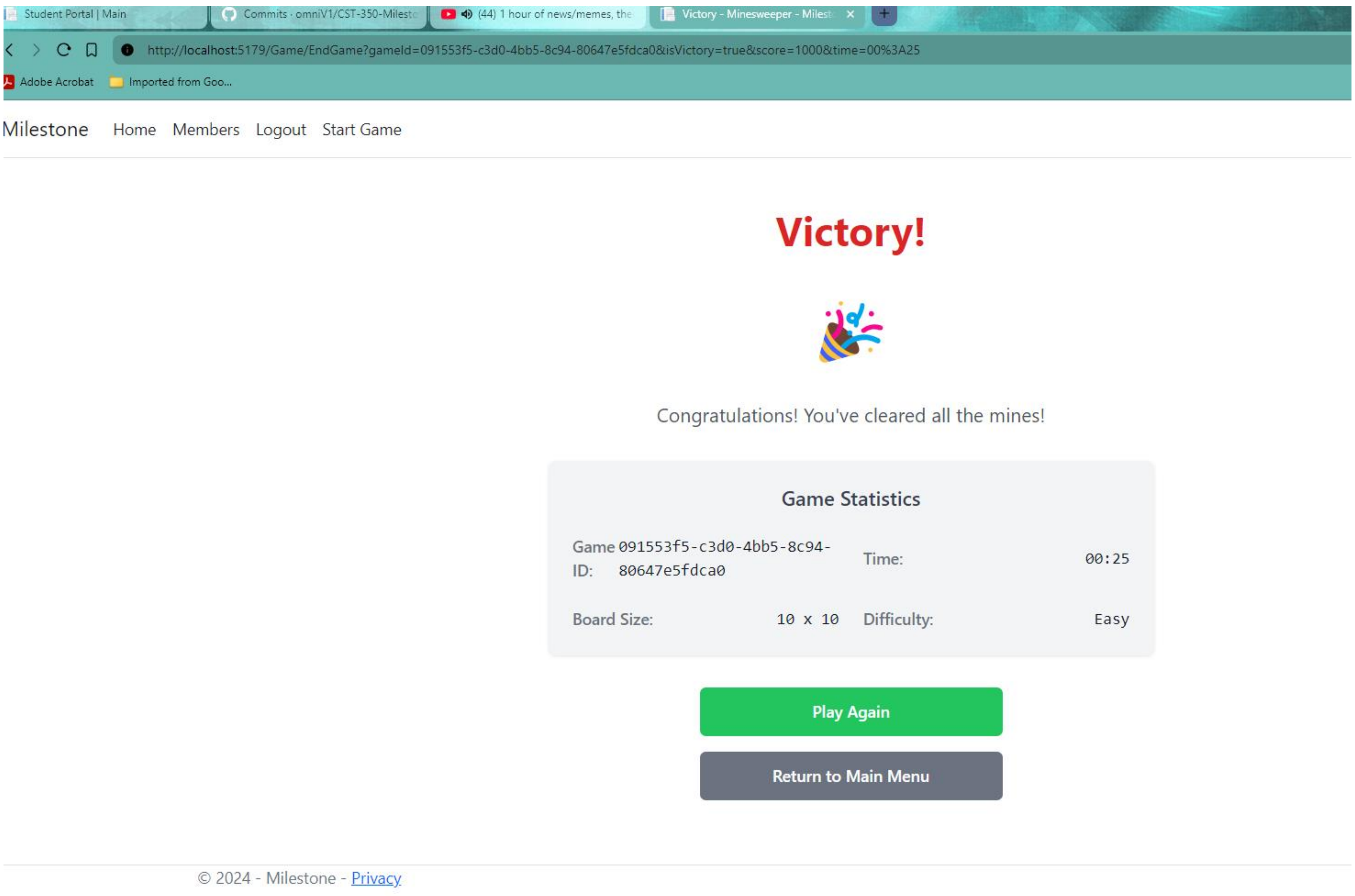


The user is able to play minesweeper! The right clicks function by setting a flag on the selected cell. If a right click is held on a reveled cell, the user can peek behind unexposed cells to see if they are potentially dangerous. The cells respond to being peeked by becoming a darker shade of grey until the user lets go of the right click on the revealed cell.



Part 2 Screenshots of application:

If the user is successful, the victory screen will appear.



Student Portal | Main · Commits · omniV1/CST-350-Milest · (44) 1 hour of news/memes, the · Victory - Minesweeper - Milest

http://localhost:5179/Game/EndGame?gameId=091553f5-c3d0-4bb5-8c94-80647e5fdca0&isVictory=true&score=1000&time=00%3A25

Adobe Acrobat · Imported from Goo...

Milestone Home Members Logout Start Game

Victory!

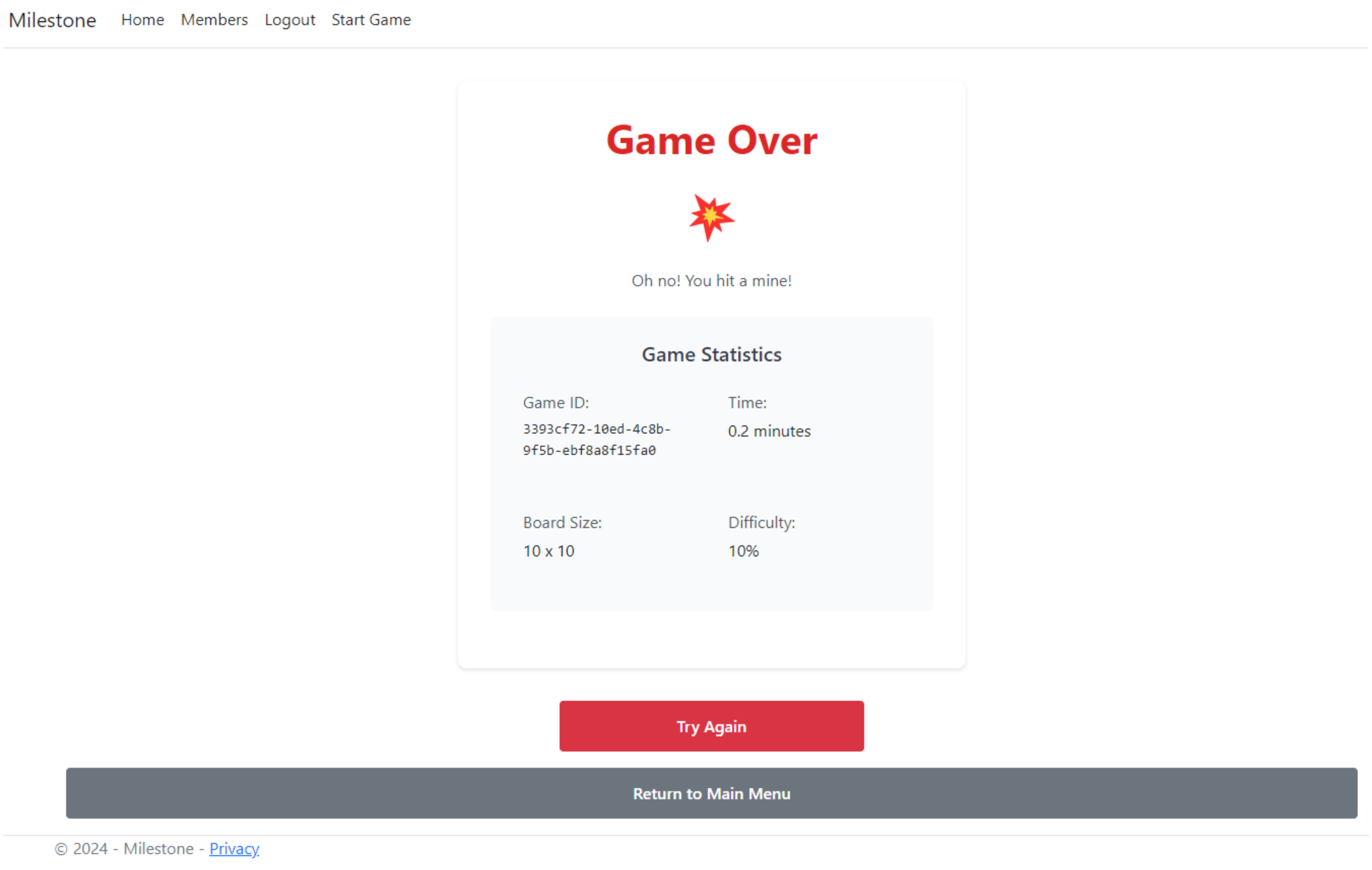
Congratulations! You've cleared all the mines!

Game Statistics

Game ID:	091553f5-c3d0-4bb5-8c94-80647e5fdca0	Time:	00:25
Board Size:	10 x 10	Difficulty:	Easy

© 2024 - Milestone - Privacy

However, if the user loses this screen appears.



Milestone Home Members Logout Start Game

Game Over

Oh no! You hit a mine!

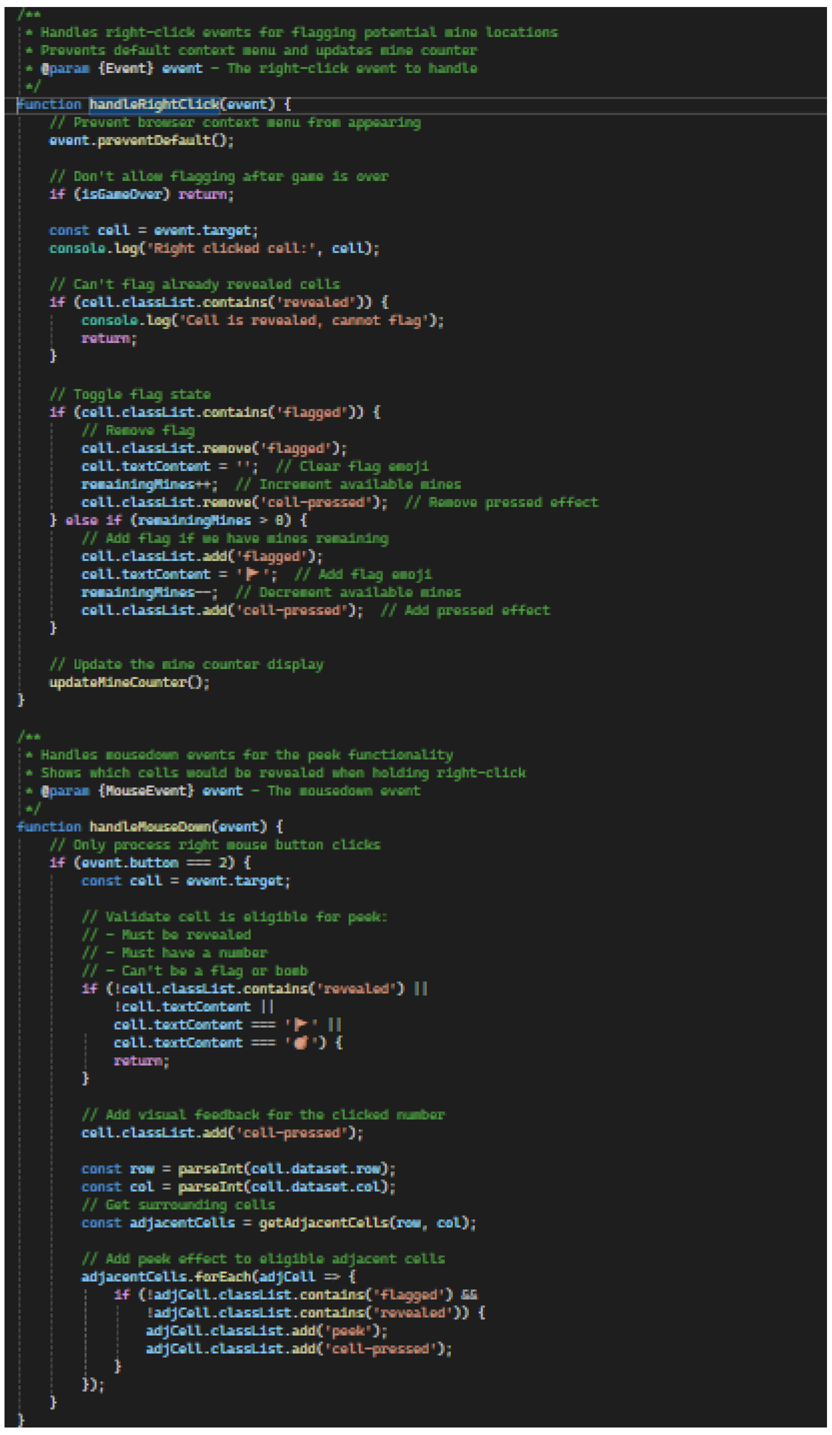
Game Statistics

Game ID:	Time:
3393cf72-10ed-4c8b-9f5b-ebf8a8f15fa0	0.2 minutes
Board Size:	Difficulty:
10 x 10	10%

© 2024 - Milestone - Privacy

Part 3 Screenshots of Functions in Application

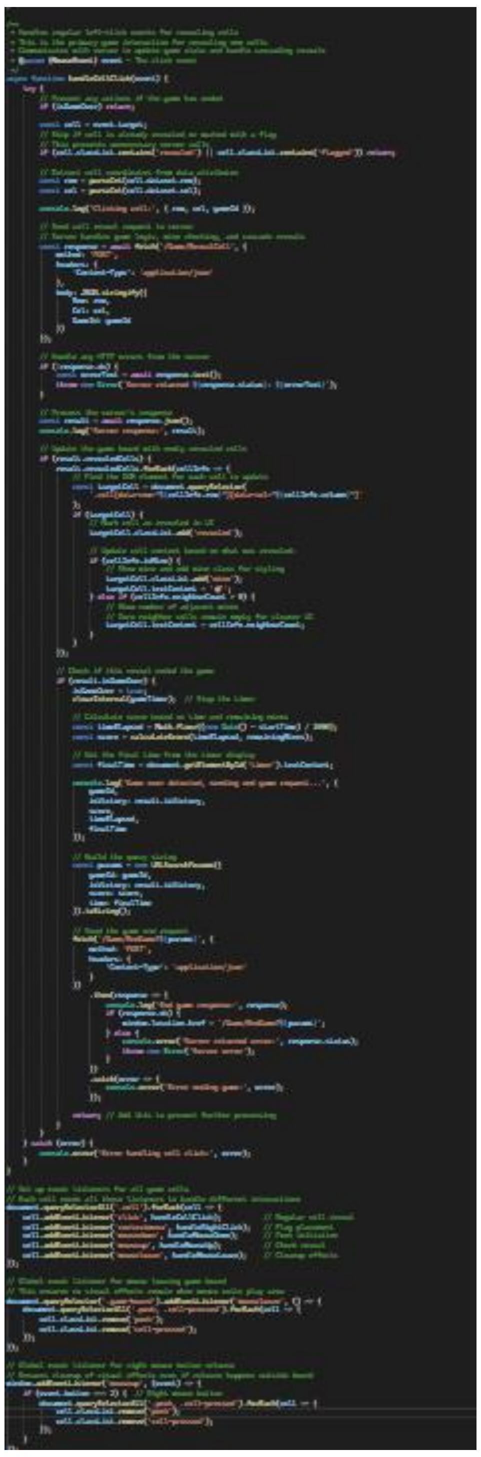
This code snippet shows the implementation of right-click event handling in the Minesweeper game, specifically the handleRightClick() and handleMouseDown() functions. The code controls how players can flag potential mine locations and implements the peek functionality when holding right-click on revealed numbers. These are client side and do not require AJAX unlike the next function.



```javascript
/**
 * Handles right-click events for flagging potential mine locations
 * Prevents default context menu and updates mine counter
 * @param {Event} event - The right-click event to handle
 */
function handleRightClick(event) {
    // Prevent browser context menu from appearing
    event.preventDefault();

    // Don't allow flagging after game is over
    if (isGameOver) return;

    const cell = event.target;
    console.log('Right clicked cell:', cell);

    // Can't flag already revealed cells
    if (cell.classList.contains('revealed')) {
        console.log('Cell is revealed, cannot flag');
        return;
    }

    // Toggle flag state
    if (cell.classList.contains('flagged')) {
        // Remove flag
        cell.classList.remove('flagged');
        cell.textContent = '';  // Clear flag emoji
        remainingMines++;  // Increment available mines
        cell.classList.remove('cell-pressed');  // Remove pressed effect
    } else if (remainingMines > 0) {
        // Add flag if we have mines remaining
        cell.classList.add('flagged');
        cell.textContent = '🚩';  // Add flag emoji
        remainingMines--;  // Decrement available mines
        cell.classList.add('cell-pressed');  // Add pressed effect
    }

    // Update the mine counter display
    updateMineCounter();
}

/**
 * Handles mousedown events for the peek functionality
 * Shows which cells would be revealed when holding right-click
 * @param {MouseEvent} event - The mousedown event
 */
function handleMouseDown(event) {
    // Only process right mouse button clicks
    if (event.button === 2) {
        const cell = event.target;

        // Validate cell is eligible for peek:
        // - Must be revealed
        // - Must have a number
        // - Can't be a flag or bomb
        if (!cell.classList.contains('revealed') ||
            !cell.textContent ||
            cell.textContent === '🚩' ||
            cell.textContent === '💣') {
            return;
        }

        // Add visual feedback for the clicked number
        cell.classList.add('cell-pressed');

        const row = parseInt(cell.dataset.row);
        const col = parseInt(cell.dataset.col);
        // Get surrounding cells
        const adjacentCells = getAdjacentCells(row, col);

        // Add peek effect to eligible adjacent cells
        adjacentCells.forEach(adjCell => {
            if (!adjCell.classList.contains('flagged') &&
                !adjCell.classList.contains('revealed')) {
                adjCell.classList.add('peek');
                adjCell.classList.add('cell-pressed');
            }
        });
    }
}
```

Part 3 Screenshots of Functions in Application

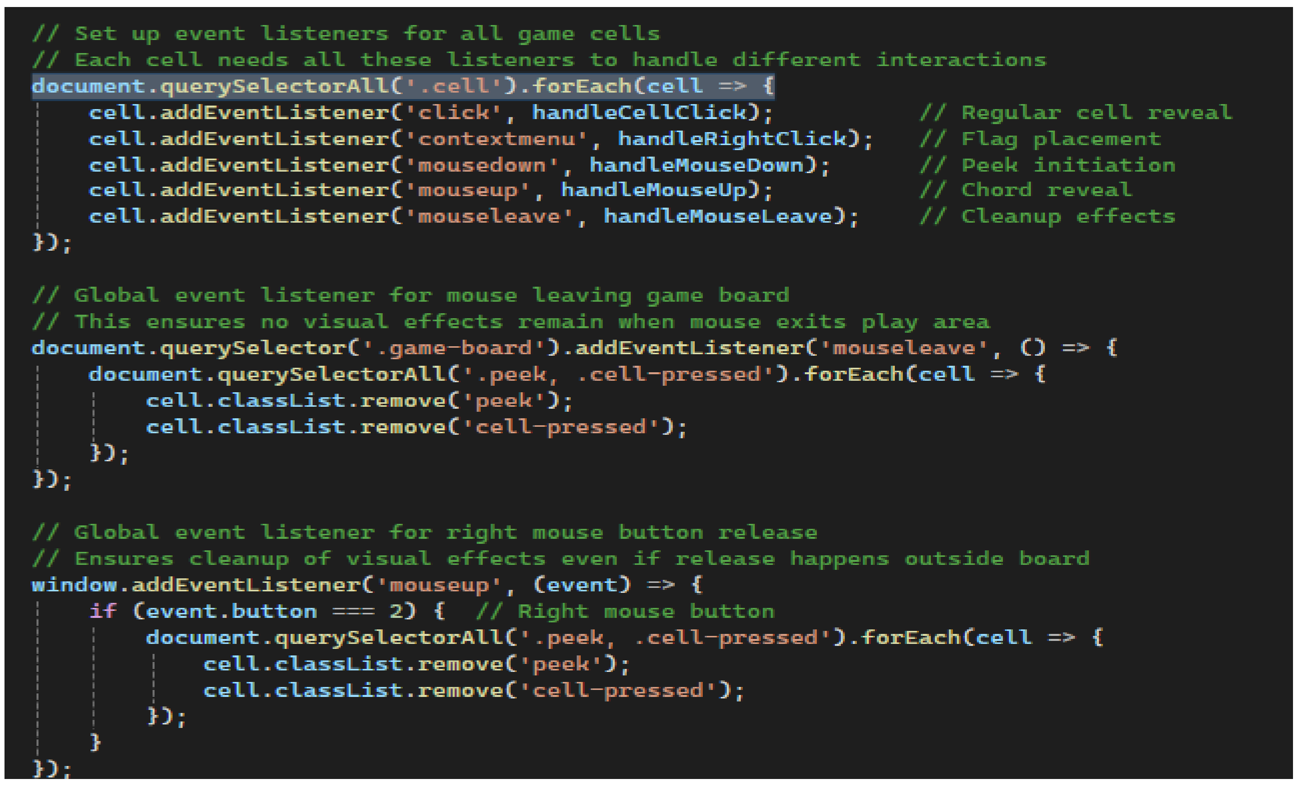
The handleCellClick function is responsible for revealing cells. It uses AJAX to send a POST request to the server with the clicked cell's coordinates, allowing the server to handle the game logic and update the game state. The function then processes the server's response, updating the UI to display the revealed cell's contents. If the server indicates the game has ended, the function calculates the player's score, sends another AJAX request to the server, and redirects the user to the game end page.



```
/**
 * Handles regular left-click events for revealing cells
 * This is the primary game interaction for revealing new cells
 * Communicates with server to update game state and handle cascading reveals
 * @param {MouseEvent} event - The click event
 */
async function handleCellClick(event) {
    try {
        // Prevent any actions if the game has ended
        if (isGameOver) return;

        const cell = event.target;
        // Skip if cell is already revealed or marked with a flag
        // This prevents unnecessary server calls
        if (cell.classList.contains('revealed') || cell.classList.contains('flagged')) return;

        // Extract cell coordinates from data attributes
        const row = parseInt(cell.dataset.row);
        const col = parseInt(cell.dataset.col);

        console.log('Clicking cell:', { row, col, gameId });

        // Send cell reveal request to server
        // Server handles game logic, mine checking, and cascade reveals
        const response = await fetch('/Game/RevealCell', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                Row: row,
                Col: col,
                GameId: gameId
            })
        });

        // Handle any HTTP errors from the server
        if (!response.ok) {
            const errorText = await response.text();
            throw new Error(`Server returned ${response.status}: ${errorText}`);
        }

        // Process the server's response
        const result = await response.json();
        console.log('Server response:', result);

        // Update the game board with newly revealed cells
        if (result.revealedCells) {
            result.revealedCells.forEach(cellInfo => {
                // Find the DOM element for each cell to update
                const targetCell = document.querySelector(
                    `.cell[data-row="${cellInfo.row}"][data-col="${cellInfo.column}"]`
                );
                if (targetCell) {
                    // Mark cell as revealed in UI
                    targetCell.classList.add('revealed');

                    // Update cell content based on what was revealed
                    if (cellInfo.isMine) {
                        // Show mine and add mine class for styling
                        targetCell.classList.add('mine');
                        targetCell.textContent = '💣';
                    } else if (cellInfo.neighbourCount > 0) {
                        // Show number of adjacent mines
                        // Zero neighbor cells remain empty for cleaner UI
                        targetCell.textContent = cellInfo.neighbourCount;
                    }
                }
            });

            // Check if this reveal ended the game
            if (result.isGameOver) {
                isGameOver = true;
                clearInterval(gameTimer); // Stop the timer

                // Calculate score based on time and remaining mines
                const timeElapsed = Math.floor((new Date() - startTime) / 1000);
                const score = calculateScore(timeElapsed, remainingMines);

                // Get the final time from the timer display
                const finalTime = document.getElementById('timer').textContent;

                console.log('Game over detected, sending end game request...', {
                    gameId,
                    isVictory: result.isVictory,
                    score,
                    timeElapsed,
                    finalTime
                });

                // Build the query string
                const params = new URLSearchParams({
                    gameId: gameId,
                    isVictory: result.isVictory,
                    score: score,
                    time: finalTime
                }).toString();

                // Send the game end request
                fetch(`/Game/EndGame?${params}`, {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json'
                    }
                })
                    .then(response => {
                        console.log('End game response:', response);
                        if (response.ok) {
                            window.location.href = `/Game/EndGame?${params}`;
                        } else {
                            console.error('Server returned error:', response.status);
                            throw new Error('Server error');
                        }
                    })
                    .catch(error => {
                        console.error('Error ending game:', error);
                    });

                return; // Add this to prevent further processing
            }
        }
    } catch (error) {
        console.error('Error handling cell click:', error);
    }
}

// Set up event listeners for all game cells
// Each cell needs all these listeners to handle different interactions
document.querySelectorAll('.cell').forEach(cell => {
    cell.addEventListener('click', handleCellClick);          // Regular cell reveal
    cell.addEventListener('contextmenu', handleRightClick);   // Flag placement
    cell.addEventListener('mousedown', handleMouseDown);      // Peek initiation
    cell.addEventListener('mouseup', handleMouseUp);          // Chord reveal
    cell.addEventListener('mouseleave', handleMouseLeave);    // Cleanup effects
});

// Global event listener for mouse leaving game board
// This ensures no visual effects remain when mouse exits play area
document.querySelector('.game-board').addEventListener('mouseleave', () => {
    document.querySelectorAll('.peek, .cell-pressed').forEach(cell => {
        cell.classList.remove('peek');
        cell.classList.remove('cell-pressed');
    });
});

// Global event listener for right mouse button release
// Ensures cleanup of visual effects even if release happens outside board
window.addEventListener('mouseup', (event) => {
    if (event.button === 2) { // Right mouse button
        document.querySelectorAll('.peek, .cell-pressed').forEach(cell => {
            cell.classList.remove('peek');
            cell.classList.remove('cell-pressed');
        });
    }
});
```

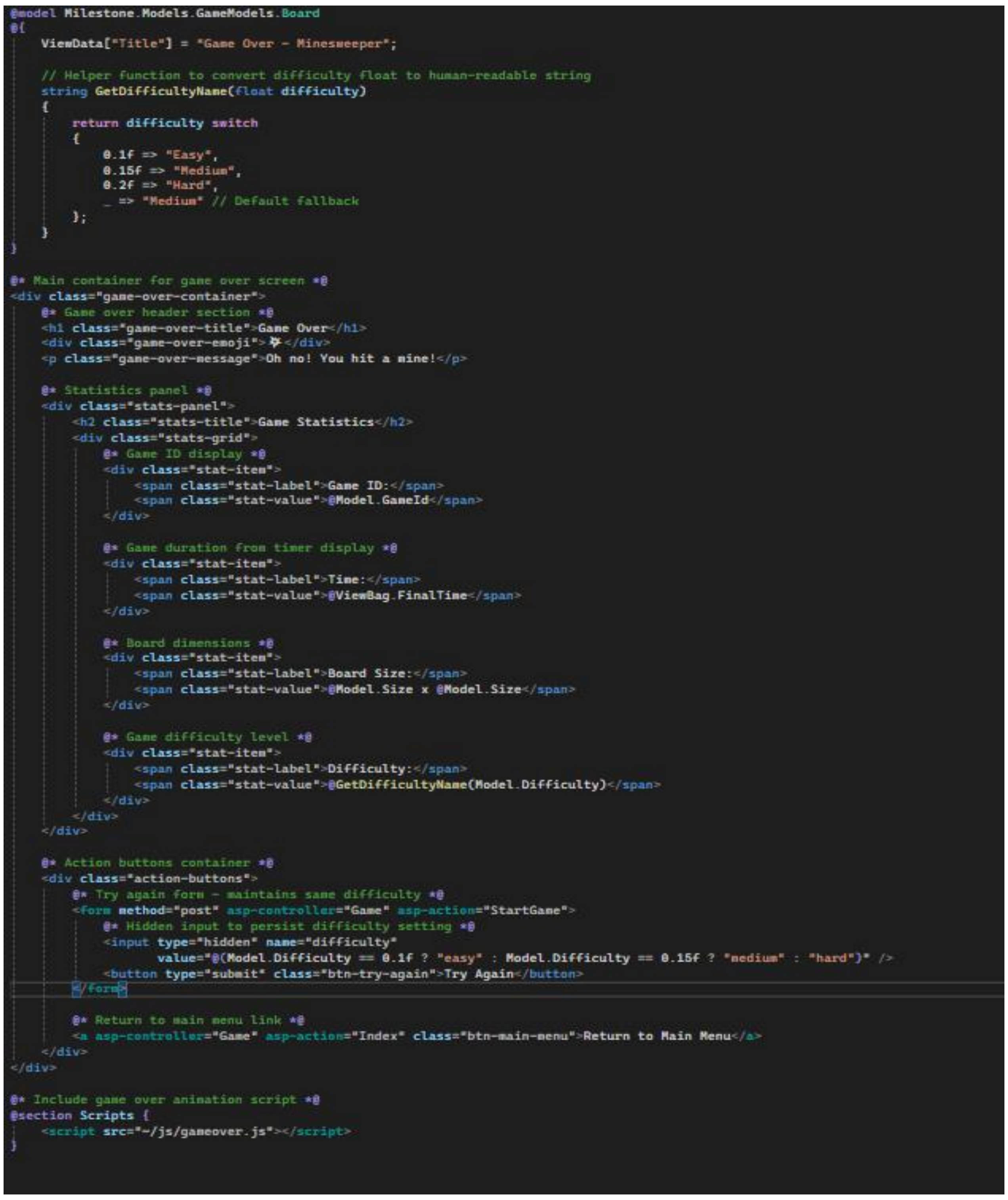
Part 3 Screenshots of Functions in Application

Sets up event listeners for various interactions on each game cell, including regular cell reveal, flag placement, peek initiation, chord reveal, and cleanup of visual effects. Adding a 'contextmenu' event listener to each cell that calls the handleRightClick function, which handles the right-click event and allows the user to place or remove flags on the cells.



```javascript
// Set up event listeners for all game cells
// Each cell needs all these listeners to handle different interactions
document.querySelectorAll('.cell').forEach(cell => {
    cell.addEventListener('click', handleCellClick);           // Regular cell reveal
    cell.addEventListener('contextmenu', handleRightClick);    // Flag placement
    cell.addEventListener('mousedown', handleMouseDown);       // Peek initiation
    cell.addEventListener('mouseup', handleMouseUp);           // Chord reveal
    cell.addEventListener('mouseleave', handleMouseLeave);     // Cleanup effects
});

// Global event listener for mouse leaving game board
// This ensures no visual effects remain when mouse exits play area
document.querySelector('.game-board').addEventListener('mouseleave', () => {
    document.querySelectorAll('.peek, .cell-pressed').forEach(cell => {
        cell.classList.remove('peek');
        cell.classList.remove('cell-pressed');
    });
});

// Global event listener for right mouse button release
// Ensures cleanup of visual effects even if release happens outside board
window.addEventListener('mouseup', (event) => {
    if (event.button === 2) {  // Right mouse button
        document.querySelectorAll('.peek, .cell-pressed').forEach(cell => {
            cell.classList.remove('peek');
            cell.classList.remove('cell-pressed');
        });
    }
});
```

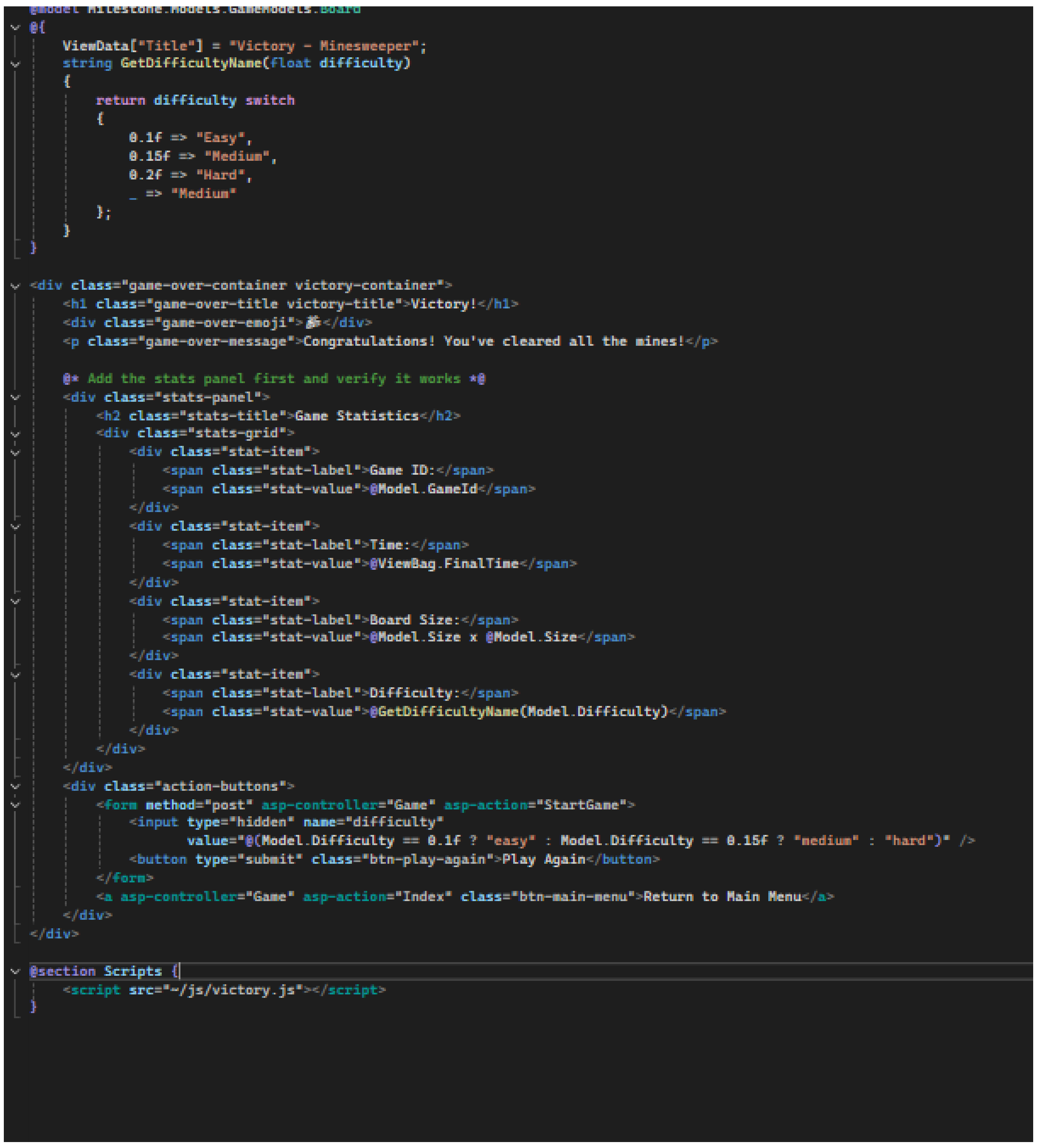
Loss view contents.



```cshtml
@model Milestone.Models.GameModels.Board
@{
    ViewData["Title"] = "Game Over - Minesweeper";

    // Helper function to convert difficulty float to human-readable string
    string GetDifficultyName(float difficulty)
    {
        return difficulty switch
        {
            0.1f => "Easy",
            0.15f => "Medium",
            0.2f => "Hard",
            _ => "Medium" // Default fallback
        };
    }
}

@* Main container for game over screen *@
<div class="game-over-container">
    @* Game over header section *@
    <h1 class="game-over-title">Game Over</h1>
    <div class="game-over-emoji">💥</div>
    <p class="game-over-message">Oh no! You hit a mine!</p>

    @* Statistics panel *@
    <div class="stats-panel">
        <h2 class="stats-title">Game Statistics</h2>
        <div class="stats-grid">
            @* Game ID display *@
            <div class="stat-item">
                <span class="stat-label">Game ID:</span>
                <span class="stat-value">@Model.GameId</span>
            </div>

            @* Game duration from timer display *@
            <div class="stat-item">
                <span class="stat-label">Time:</span>
                <span class="stat-value">@ViewBag.FinalTime</span>
            </div>

            @* Board dimensions *@
            <div class="stat-item">
                <span class="stat-label">Board Size:</span>
                <span class="stat-value">@Model.Size x @Model.Size</span>
            </div>

            @* Game difficulty level *@
            <div class="stat-item">
                <span class="stat-label">Difficulty:</span>
                <span class="stat-value">@GetDifficultyName(Model.Difficulty)</span>
            </div>
        </div>
    </div>

    @* Action buttons container *@
    <div class="action-buttons">
        @* Try again form - maintains same difficulty *@
        <form method="post" asp-controller="Game" asp-action="StartGame">
            @* Hidden input to persist difficulty setting *@
            <input type="hidden" name="difficulty"
                   value="@(Model.Difficulty == 0.1f ? "easy" : Model.Difficulty == 0.15f ? "medium" : "hard")" />
            <button type="submit" class="btn-try-again">Try Again</button>
        </form>

        @* Return to main menu link *@
        <a asp-controller="Game" asp-action="Index" class="btn-main-menu">Return to Main Menu</a>
    </div>
</div>

@* Include game over animation script *@
@section Scripts {
    <script src="~/js/gameover.js"></script>
}
```

Part 3 Screenshots of Functions in Application

Win view contents.



```
@model Milestone.Models.GameModels.Board
@{
    ViewData["Title"] = "Victory - Minesweeper";
    string GetDifficultyName(float difficulty)
    {
        return difficulty switch
        {
            0.1f => "Easy",
            0.15f => "Medium",
            0.2f => "Hard",
            _ => "Medium"
        };
    }
}

<div class="game-over-container victory-container">
    <h1 class="game-over-title victory-title">Victory!</h1>
    <div class="game-over-emoji">🎉</div>
    <p class="game-over-message">Congratulations! You've cleared all the mines!</p>

    @* Add the stats panel first and verify it works *@
    <div class="stats-panel">
        <h2 class="stats-title">Game Statistics</h2>
        <div class="stats-grid">
            <div class="stat-item">
                <span class="stat-label">Game ID:</span>
                <span class="stat-value">@Model.GameId</span>
            </div>
            <div class="stat-item">
                <span class="stat-label">Time:</span>
                <span class="stat-value">@ViewBag.FinalTime</span>
            </div>
            <div class="stat-item">
                <span class="stat-label">Board Size:</span>
                <span class="stat-value">@Model.Size x @Model.Size</span>
            </div>
            <div class="stat-item">
                <span class="stat-label">Difficulty:</span>
                <span class="stat-value">@GetDifficultyName(Model.Difficulty)</span>
            </div>
        </div>
    </div>
    <div class="action-buttons">
        <form method="post" asp-controller="Game" asp-action="StartGame">
            <input type="hidden" name="difficulty"
                   value="@(Model.Difficulty == 0.1f ? "easy" : Model.Difficulty == 0.15f ? "medium" : "hard")" />
            <button type="submit" class="btn-play-again">Play Again</button>
        </form>
        <a asp-controller="Game" asp-action="Index" class="btn-main-menu">Return to Main Menu</a>
    </div>
</div>

@section Scripts {
    <script src="~/js/victory.js"></script>
}
```

Part 4 Summary of Key Concepts:

Rather than refreshing the entire game page with each user interaction, I have incorporated AJAX to enable partial page updates. This allows the game board to dynamically update individual cells as they are revealed or flagged, providing a more responsive and efficient user experience. To demonstrate the use of AJAX, I have added a timestamp display on the game screen to show that the updates are happening in real-time without a full page reload.

Building on the previous implementation, I have implemented a JavaScript (or jQuery) right-click event on each game cell. This right-click event allows the user to plant a flag or remove an existing flag on the selected cell. Additionally, I have ensured that a cell with a flag will no longer respond to left-click events, preventing unintended reveals.