***Rapport***

The provided JavaScript code is a comprehensive game script for a turn-based strategy game involving two players, AI opponents, machine learning (ML) predictions, and animations. Below is a detailed analysis of the roles, connections between functions, and classifications based on their purposes.

### 🌟 ****Overall Structure****

This code handles:

* Game logic
* UI interactions
* Animations
* AI behavior using Minimax algorithm
* ML combat prediction using **brain.js**
* Saving/loading game state and statistics

It uses classes like **Unit** and **Clan** from external files (**units.js**) and imports animations from **animations.js**.

## 🔍 ****Class Analysis****

### 1. Unit

* **Role** : Represents individual units with stats like health, damage, defense, level.
* **Used in** : Game logic, movement, attack phases, ML predictions.
* **Connection** : Used by **renderUnit**, **performAttack**, **gainExperience**, and ML functions.

### 2. Clan

* **Role** : Defines clans with specific unit generation logic, bonuses, and visuals.
* **Used in** : Setup phase, unit placement, clan selection, visual rendering.
* **Connection** : Used by **generateUnitsToPlace**, **startGame**, and victory screen.

## 🧠 ****Machine Learning (ML) Functions****

| **Function Name** | **Role** | **Connected With** | **Description** |
| --- | --- | --- | --- |
| **initializeNeuralNetwork()** | Initializes neural network for combat prediction | **generateInitialCombatData()**,**trainCombatModel()** | Sets up the brain.js model |
| **generateInitialCombatData()** | Generates sample data for training | **initializeNeuralNetwork()** | Provides mock battles for training |
| **trainCombatModel()** | Trains the neural network with combat history | **recordCombatResult()** | Uses past results to improve predictions |
| **predictCombatOutcome(attacker, defender)** | Predicts battle outcome | **displayPrediction()** | Returns win/loss probability |
| **recordCombatResult(attacker, defender, result)** | Records real battle outcomes | **performAttack()** | Feeds real data into the ML model |
| **addMLVisualization()** | Adds UI element for ML predictions | **displayPrediction()** | Shows ML insights in sidebar |
| **addMLStyles()** | Styles for ML UI | **addMLVisualization()** | CSS styling for prediction display |
| **analyzeCombatTrends()** | Analyzes historical combat data | **showCombatInsights()** | Gives statistical insights |
| **addCombatInsightsButton()** | Adds button to show trends | **showCombatInsights()** | UI trigger for combat analysis |

**🤖 Artificial Intelligence (AI) Functions**

| **Function Name** | **Role** | **Connected With** | **Description** |
| --- | --- | --- | --- |
| **implementAI()** | Sets up AI controls and difficulty selector | **toggleAI()** | Enables AI opponent toggle |
| **toggleAI()** | Toggles AI on/off | **aiTakeTurn()** | Starts AI's turn |
| **aiTakeTurn()** | Main AI decision loop | **aiMovementPhase()**,**aiActionPhase()** | Controls AI during its turn |
| **aiMovementPhase()** | AI chooses where to move | **findBestMoveWithMinimax()** | Decides best movement position |
| **aiActionPhase()** | AI chooses action (attack or end turn) | **performAttack()** | Decides to attack or skip |
| **findUnitThatCanAttack(units)** | Finds unit that can attack | **aiMovementPhase()** | Helps AI prioritize attacking units |
| **findMoveTowardEnemy(unitInfo, movablePositions)** | Moves toward nearest enemy | **aiMovementPhase()** | Basic AI pathing |
| **findLowestHealthTarget(attackablePositions)** | Targets weakest enemy | **aiActionPhase()** | Prioritizes low-health targets |
| **findBestUnitWithMinimax(units)** | Evaluates best unit to act | **evaluateUnitPotential()** | Advanced unit choice |
| **evaluateUnitPotential(unitInfo)** | Scores unit's usefulness | **findBestUnitWithMinimax()** | Heuristic for Minimax |
| **findBestMoveWithMinimax(unitInfo, movablePositions, depth)** | Best possible move | **aiMovementPhase()** | Smart movement decisions |
| **findBestTargetWithMinimax(attackablePositions, depth)** | Best target to attack | **aiActionPhase()** | Tactical targeting |

## 🎮 ****User Interface (UI) Functions****

| **Function Name** | **Role** | **Connected With** | **Description** |
| --- | --- | --- | --- |
| **createGameBoard()** | Creates board grid | DOM | Renders the 10x10 board |
| **updateUI()** | Updates player info, buttons, phase | **getPhaseText()**,**updateButtons()** | Syncs UI with game state |
| **getPhaseText(phase)** | Converts phase enum to text | **updateUI()** | Displays readable phase name |
| **updateButtons()** | Disables/enables UI buttons | **gameState.phase** | Ensures correct button states |
| **updateUnitCounts()** | Shows remaining units per player | **generateUnitsToPlace()** | Reflects current unit count |
| **handleCellClick(row, col)** | Handles user clicks on board | **handlePlacementPhase()**,  **handleMovementPhase()**,**handleActionPhase()** | Central click handler |
| **renderUnit(row, col, unit)** | Renders unit image and health | **handleCellClick()** | Places unit visually |
| **logMessage(message)** | Logs game events | All major actions | Keeps track of game history |
| **clearSelection()** | Clears selected unit | **handleCellClick()** | Deselects unit after action |
| **clearHighlightedCells()** | Removes highlight effects | **clearSelection()** | Visual cleanup |
| **clearCell(row, col, numero)** | Removes unit from board | **performAttack()** | Clears defeated unit |
| **displayGameOver(winner)** | Shows victory screen | **checkWinCondition()** | Final screen with winner |
| **saveGameResult(winner, gameState)** | Saves game result | **displayGameOver()** | Stores in localStorage |
| **displayGameHistory()** | Shows game history | **saveGameResult()** | Retrieves and displays saved games |

## 🎥 ****Animation Functions****

| **Function Name** | **Role** | **Connected With** | **Description** |
| --- | --- | --- | --- |
| **rollDiceForInitiative()** | Dice roll animation | **attaqueDiceForInitiative()** | Randomly decides who starts |
| **attaqueDiceForInitiative()** | Dice roll for attack success | **performAttack()** | Determines initiative |
| **prepareAnimationElements()** | Prepares dice for animation | **rollDiceForInitiative()** | Adds animation class |
| **highlightWinner(roll1, roll2)** | Highlights winning dice | **rollDiceForInitiative()** | Visual feedback |
| **disableControls()**/**enableControls()** | Lock/unlock UI during animation | **rollDiceForInitiative()** | Prevents input during  animations |
| **playDiceSound()** | Plays dice sound effect | **rollDiceForInitiative()** | Enhances experience |
| **animateDefenseEffect(row, col)** | Defense animation | **defendButton** | Visual indicator when  defending |
| **animateBlockEffect(row, col)** | Block animation | **performAttack()** | Visual block effect on defense |
| **animations.animateAttack()** | Attack animation | **performAttack()** | Movement + hit animation |
| **animations.showDamageNumber()** | Shows floating damage number | **performAttack()** | Visual damage feedback |

## ⚙️ ****Game Logic Functions****

| **Function Name** | **Role** | **Connected With** | **Description** |
| --- | --- | --- | --- |
| **startGame()** | Begins game after clan selection | **generateUnitsToPlace()** | Initializes board and units |
| **generateUnitsToPlace()** | Generates units based on clan | **startGame()** | Loads unit list for placement |
| **handlePlacementPhase(row, col)** | Unit placement logic | **handleCellClick()** | Allows placing units |
| **handleMovementPhase(row, col)** | Handles unit movement | **handleCellClick()** | Move unit to adjacent cell |
| **handleActionPhase(row, col)** | Handles attack or defend | **handleCellClick()** | Action phase logic |
| **performAttack(attackingUnit, defendingUnit, defRow, defCol)** | Core attack function | **handleActionPhase()** | Damages enemy, checks death |
| **gainExperience(unit, damage)** | Grants XP on successful hit | **performAttack()** | Increases unit level |
| **levelUp(unit)** | Levels up unit and increases stats | **gainExperience()** | Stats increase, visual effect |
| **checkWinCondition()** | Checks if any player won | **performAttack()** | Ends game if no units left |
| **prepareNextTurn()** | Switches to next player | **endTurnButton**,**aiTakeTurn()** | Advances game state |
| **checkAndRemoveDefenseBonus()** | Removes defense bonus at start of new round | **prepareNextTurn()** | Resets defensive stance |

| **Function Name** | **Role** | **Connected With** | **Description** |
| --- | --- | --- | --- |
| **loadPreviousCombatHistory()** | Loads saved ML data | **initializeNeuralNetwork()** | Restores past battle records |
| **initializePlayerPerformance()** | Initializes performance tracking | **gameState.playerPerformance** | Tracks wins, losses, etc. |
| **updateAIDifficulty()** | Adjusts AI difficulty based on performance | **initializePlayerPerformance()** | Dynamic difficulty scaling |
| **updateGameStatistics(winner)** | Updates game stats after match | **displayGameOver()** | Saves win/loss, updates AI |
| **saveGameResult(winner, gameState)** | Saves match result to localStorage | **displayGameOver()** | For future history display |
| **displayGameHistory()** | Shows saved match history | **saveGameResult()** | UI for viewing previous matches |

## 💾 ****Data Handling & Statistics****

| **Function Name** | **Role** | **Connected With** | **Description** |
| --- | --- | --- | --- |
| **window.onload = initializeAIAndML;** | Entry point | All above functions | Starts the game |
| **normalize(val, min, max)** | Normalizes values for ML | **predictCombatOutcome()** | Converts to 0-1 scale |
| **document.addEventListener("DOMContentLoaded", function () { ... })** | Ensures DOM is ready | All | Waits until HTML is loaded |

## 🌲 ****Function Call Tree (Simplified)****

window.onload

└── initializeAIAndML()

├── loadPreviousCombatHistory()

├── initializePlayerPerformance()

├── initializeNeuralNetwork()

│ └── generateInitialCombatData()

│ └── trainCombatModel()

├── addMLVisualization()

├── addMLStyles()

├── implementAI()

│ └── toggleAI()

│ └── aiTakeTurn()

│ ├── aiMovementPhase()

│ │ ├── findBestMoveWithMinimax()

│ │ └── animateUnitMovement()

│ └── aiActionPhase()

│ └── performAttack()

│ ├── predictCombatOutcome()

│ ├── recordCombatResult()

│ ├── gainExperience()

│ ├── checkWinCondition()

│ └── animateAttack()

├── createGameBoard()

├── startButton.onclick

│ └── startGame()

│ └── generateUnitsToPlace()

│ └── updateUI()

├── handleCellClick()

│ ├── handlePlacementPhase()

│ ├── handleMovementPhase()

│ └── handleActionPhase()

│ └── performAttack()

├── logMessage()

├── displayGameOver()

├── saveGameResult()

├── displayGameHistory()

└── ...

## ✅ ****Summary Table: Function Classifications****

| **Category** | **Functions** |
| --- | --- |
| **ML** | **initializeNeuralNetwork**,**trainCombatModel**,**predictCombatOutcome**,  **recordCombatResult**,**analyzeCombatTrends**,**addMLVisualization**,**addMLStyles**,**displayPrediction** |
| **AI** | **implementAI**,**toggleAI**,**aiTakeTurn**,**aiMovementPhase**,**aiActionPhase**,**findUnitThatCanAttack**,  **findMoveTowardEnemy**,**findLowestHealthTarget**,**findBestUnitWithMinimax**,**evaluateUnitPotential**,  **findBestMoveWithMinimax**,**findBestTargetWithMinimax** |
| **UI** | **createGameBoard**,**updateUI**,**getPhaseText**,**updateButtons**,**updateUnitCounts**,**handleCellClick**,  **renderUnit**,**logMessage**,**clearSelection**,**clearHighlightedCells**,**clearCell**,**displayGameOver**,  **saveGameResult**,**displayGameHistory** |
| **Animations** | **rollDiceForInitiative**,**attaqueDiceForInitiative**,**prepareAnimationElements**,**highlightWinner**,  **disableControls**,**enableControls**,**playDiceSound**,**animateDefenseEffect**,**animateBlockEffect**,  **animations.animateAttack**,**animations.showDamageNumber** |
| **Game Logic** | **startGame**,**generateUnitsToPlace**,**handlePlacementPhase**,**handleMovementPhase**,**handleActionPhase**,  **performAttack**,**gainExperience**,**levelUp**,**checkWinCondition**,**prepareNextTurn**,  **checkAndRemoveDefenseBonus** |
| **Data & Stats** | **loadPreviousCombatHistory**,**initializePlayerPerformance**,**updateAIDifficulty**,  **updateGameStatistics**,**saveGameResult**,**displayGameHistory** |

## 🧩 ****Game Architecture Diagram (Text-Based)****

+---------------------------------------------------------------+

| Game Entry Point |

| window.onload = initializeAIAndML |

+--------------------------+------------------------------------+

|

+------------------+---------------------+

| |

+--------v------+ +-----------v-------------+

| Load History | | Initialize Player Stats |

+----------------+ +-------------------------+

|

v

+--------v-------+

| Initialize ML |<---+ Generate Initial Combat Data

+--------+--------+ |

| |

v v

+--------+--------+ +--v-----------------+

| Train NeuralNet | | Add ML UI/Styles |

+--------+--------+ +--------------------+

|

v

+--------v--------+

| Implement AI |<--- Select Difficulty

+--------+--------+

|

v

+--------v--------+

| Create Game Board|

+--------+--------+

|

v

+--------v--------+

| Start Button |<--- Clan Selection

+--------+--------+

|

v

+--------v--------+

| Start Game |<--- Unit Placement Phase

+--------+--------+

|

v

+--------v--------+

| Dice Roll |<--- Determine Initiative

+--------+--------+

|

## 🔍 ****Detailed Breakdown by Module****

### 1. ****Initialization****

* **Functions** : **window.onload**, **initializeAIAndML**, **loadPreviousCombatHistory**, **initializePlayerPerformance**
* **Purpose** : Loads saved stats, initializes player performance tracking, and sets up the neural network.
* **Used In** : Game setup before any action starts.

### 2. ****Machine Learning (ML) System****

* **Core Functions** : **initializeNeuralNetwork**, **trainCombatModel**, **predictCombatOutcome**, **recordCombatResult**
* **Data Source** : Uses combat history from past battles.
* **Prediction Use** : Used during attacks to predict outcome (win/loss).
* **UI Integration** : **displayPrediction()** shows predictions live on screen.
* **Visuals** : **addMLVisualization()**, **addMLStyles()** for real-time feedback.

### 3. ****Artificial Intelligence (AI) System****

* **Core Functions** : **implementAI**, **toggleAI**, **aiTakeTurn**, **aiMovementPhase**, **aiActionPhase**
* **Difficulty Levels** : Easy (random), Medium (heuristic), Hard (Minimax)
* **Minimax Logic** : **findBestUnitWithMinimax**, **evaluateUnitPotential**, **findBestMoveWithMinimax**, **findBestTargetWithMinimax**
* **AI Decisions** :
  + Move toward enemy
  + Choose best unit
  + Target weakest enemy
* **Integration** : Automatically plays when enabled and it's AI’s turn.

### 4. ****User Interface (UI) System****

* **Board Management** : **createGameBoard**, **renderUnit**, **clearCell**, **clearSelection**
* **Buttons & Events** : **startButton**, **moveButton**, **attackButton**, **defendButton**, **endTurnButton**
* **State Updates** : **updateUI**, **getPhaseText**, **updateButtons**, **updateUnitCounts**
* **Log Messages** : **logMessage** for tracking actions.

### 5. ****Animations & Effects****

* **Dice Rolling** : **rollDiceForInitiative**, **attaqueDiceForInitiative**, **prepareAnimationElements**, **highlightWinner**
* **Attack Effects** : **animateAttack**, **showDamageNumber**, **animateDefenseEffect**, **animateBlockEffect**
* **Sounds** : Dice roll, block, victory.
* **CSS Effects** : Hit flash, death fade, defense glow, damage pop-up.

### 6. ****Game Logic****

* **Phases** : Setup → Placement → Dice Roll → Movement → Action → Game Over
* **Turn Flow** : **handleCellClick** → **handlePlacementPhase** / **handleMovementPhase** / **handleActionPhase**
* **Attack Logic** : **performAttack** includes damage calculation, XP gain, level up.
* **Victory Detection** : **checkWinCondition** → **displayGameOver**

### 7. ****Combat & Experience System****

* **Level Up** : **levelUp** increases stats, triggers visual effect.
* **XP Gain** : **gainExperience** based on damage dealt.
* **Defensive Stance** : Temporary bonus with visual indicator.

### 8. ****Game State Management****

* **Game State Object** : Tracks board, players, units, phase, etc.
* **Save/Load** : **saveGameResult**, **displayGameHistory**
* **Local Storage** : Stores results, combat history, player stats.

## 📊 ****Functional Tree (Code Call Hierarchy)****

window.onload

└── initializeAIAndML()

├── loadPreviousCombatHistory()

├── initializePlayerPerformance()

├── initializeNeuralNetwork()

│ └── generateInitialCombatData()

│ └── trainCombatModel()

├── addMLVisualization()

├── addMLStyles()

├── implementAI()

│ └── toggleAI()

│ └── aiTakeTurn()

│ ├── aiMovementPhase()

│ │ ├── findBestMoveWithMinimax()

│ │ └── animateUnitMovement()

│ └── aiActionPhase()

│ └── performAttack()

│ ├── predictCombatOutcome()

│ ├── recordCombatResult()

│ ├── gainExperience()

│ ├── checkWinCondition()

│ └── animateAttack()

├── createGameBoard()

├── startButton.onclick

│ └── startGame()

│ └── generateUnitsToPlace()

│ └── updateUI()

├── handleCellClick()

│ ├── handlePlacementPhase()

│ ├── handleMovementPhase()

│ └── handleActionPhase()

│ └── performAttack()

├── logMessage()

├── displayGameOver()

├── saveGameResult()

├── displayGameHistory()

└── ...

## 🖼️ ****Exported as Visual Diagram (Suggested Tools)****

You can export this structure into a diagram using:

### ✅ Recommended Tools:

* **Lucidchart** – Great for interactive diagrams
* **Draw.io (diagrams.net)** – Free and integrates with Google Drive
* **Mermaid Live Editor** – For markdown-based diagrams
* **PlantUML** – If you want code-based UML diagrams

### C:\Users\HP PC\Downloads\mermaid-diagram-2025-05-29-103148.png🧪 Mermaid Example: