CS390 - Mini Project 1

Tanay Gondil

Due: September 27th, 2025

Assignment Overview

Create a fully functional Wordle clone by implementing core game logic in JavaScript. You will work with provided HTML, CSS, and starter code to build an interactive word-guessing game with animations and user feedback.

1 Learning Objectives

By completing this assignment, you will demonstrate proficiency in:

- **DOM Manipulation**: Dynamically updating HTML elements with JavaScript
- **Event Handling**: Responding to keyboard and mouse interactions
- **Game Logic**: Implementing complex conditional logic and state management
- **Array & String Operations**: Processing and comparing text data
- **Function Design**: Breaking problems into manageable, reusable functions
- **UI Feedback**: Providing clear visual feedback to players
- **Data Validation**: Ensuring user input meets game requirements

2 Game Rules & Mechanics

2.1 Basic Wordle Rules

1. Players have **6 attempts** to guess a **5-letter word**
2. Each guess must be a valid English word
3. After each guess, letters are color-coded:
   - Green: Letter is correct and in the right position
   - Yellow: Letter is in the word but wrong position
   - Gray: Letter is not in the word
4. The on-screen keyboard updates to show letter states
5. Game ends when word is guessed or 6 attempts are used

2.2 Advanced Mechanics

- **Duplicate Letters**: Handle multiple instances of the same letter correctly
- **Visual Effects**: Simple celebrations and feedback
- **Statistics**: Track games played, win percentage, and streaks
- **Input Validation**: Prevent invalid inputs and provide feedback

3 Implementation Requirements

3.1 Core Game Functions (60 points)

Function	**Points**	**Requirements**
`initializeGame()`	10	Reset game state, select random word, clear board
`handleKeyPress()`	15	Process keyboard input (letters, Enter, Backspace)
`submitGuess()`	20	Validate guess, check letters, update display
`checkLetter()`	10	Implement Wordle letter-checking logic
`updateGameState()`	5	Handle win/lose conditions, update variables

Table 1: Core Function Requirements

3.1.1 Function Details

`initializeGame()` **Goal:** Set up a fresh game ready for the player to start guessing.

- Reset all tracking variables (`currentWord`, `currentGuess`, `currentRow`, etc.)
- Pick a new target word using `WordleWords.getRandomWord()`
- Clear the visual game board using the provided `resetBoard()` function
- Hide any leftover messages or modals from previous games

Think of this as the "New Game" button functionality.

`handleKeyPress(key)` **Goal:** Process what happens when a player types on their keyboard or clicks the on-screen keys.

- **Letter keys (A-Z):** Add the letter to the current guess if there's room (less than 5 letters)
- **ENTER key:** Try to submit the current guess if it's complete (exactly 5 letters)
- **BACKSPACE key:** Remove the last letter from the current guess if there are any letters
- Update the visual display of tiles after each change

This is like being the "translator" between user input and game actions.

`submitGuess()` **Goal:** Process a complete 5-letter guess and give feedback to the player.

- First, check if the guess is a real English word using `WordleWords.isValidWord()`
- If invalid, show an error message and shake the row
- If valid, compare each letter to the target word using `checkLetter()`
- Update tile colors (green/yellow/gray) and keyboard colors based on results
- Apply colors immediately to provide instant feedback
- Check if the player won or lost, then move to the next row

This is the "heart" of Wordle - where guesses get evaluated.

`checkLetter(letter, position, target)` **Goal:** Determine the color (status) of a single letter in a guess.

- Return `'correct'` if the letter matches the target word at this exact position (green)
- Return `'present'` if the letter exists in the target word but at a different position (yellow)
- Return `'absent'` if the letter doesn't exist anywhere in the target word (gray)
- **Challenge:** Handle duplicate letters correctly (e.g., if target is "SPEED" and guess is "ERASE", how many E's should be yellow vs gray?)

This implements the core Wordle logic that everyone recognizes.

`updateGameState(isCorrect)` **Goal:** Decide if the game should continue or end after a guess.

- If the guess was correct, set `gameWon = true` and `gameOver = true`
- If this was the 6th guess and still wrong, set `gameOver = true`
- Show the appropriate end-game modal with results
- Update game statistics (wins, streaks, etc.)

This handles the "What happens next?" decision after each guess.

3.2 Advanced Features (30 points)

Function	**Points**	**Requirements**
`updateKeyboardColors()`	10	Update keyboard with color priority
`processRowReveal()`	5	Handle row completion effects (simplified)
`showEndGameModal()`	10	Display results and statistics
`validateInput()`	5	Prevent invalid actions and edge cases

Table 2: Advanced Feature Requirements

3.2.1 Advanced Function Details

`updateKeyboardColors(guess, results)` **Goal:** Keep the on-screen keyboard updated with color hints from previous guesses.

- Loop through each letter in the guess and its corresponding result
- Update the keyboard key color using the provided `updateKeyboardKey()` function
- Remember: colors have priority (green ¿ yellow ¿ gray) - don't downgrade a key
- This gives players visual feedback about which letters they've tried

This creates the helpful "memory" effect where used letters stay colored.

`processRowReveal(rowIndex, results)` **Goal:** Handle any special effects when a row is completed (simplified, no animations).

- Check if the guess was completely correct (all results are 'correct')
- If so, trigger a celebration effect using the provided `celebrateRow()` function
- Focus on the core logic rather than visual effects

This handles the "you won!" celebration moment.

`showEndGameModal(won, targetWord)` **Goal:** Display the game results when the player wins or loses.

- Update the game statistics using the provided `updateStats()` function
- Show the modal with appropriate win/lose message
- Display the target word so players can see what it was
- Calculate and show how many guesses were used (if won)
- Use the provided `showModal()` function with proper parameters

This creates the final "reveal" and gives players their results.

`validateInput(key, currentGuess)` **Goal:** Prevent invalid actions before they cause problems.

- Check if the game is already over (no input allowed)
- For letter keys: ensure the current guess isn't already full (5 letters)
- For ENTER key: ensure the current guess is complete (exactly 5 letters)
- For BACKSPACE key: ensure there are letters to remove
- Return `true` if the input is valid, `false` otherwise

This prevents confusing situations and makes the game feel polished.

4 Technical Specifications

4.1 Available Resources

You have access to the following pre-built functionality:

- **DOM Elements**: All game tiles, keyboard keys, and UI components
- **Utility Functions**: `getTile()`, `updateTileDisplay()`, `setTileState()`, etc.
- **Word Management**: `WordleWords.getRandomWord()`, `WordleWords.isValidWord()`
- **Effect Helpers**: `shakeRow()`, `celebrateRow()`
- **State Variables**: `currentWord`, `currentGuess`, `currentRow`, etc.

4.2 Key Algorithms

4.2.1 Letter Checking Algorithm

The most complex part of Wordle is handling duplicate letters correctly:

```
// Pseudocode for handling duplicates
function checkLetter(guess, position, target) {
    if (target[position] === guess[position]) {
        return 'correct';
    }

    // Count available instances of this letter
    // (not already marked as correct)

    if (letterAvailable) {
        return 'present';
    }

    return 'absent';
}
```

4.2.2 Game Flow Sequence

Game actions must be properly sequenced:

1. User submits guess
2. Validate word (shake if invalid)
3. Update tile colors immediately
4. Update keyboard colors
5. Check win/lose conditions
6. Show celebration or end-game modal

5 Implementation Examples & Scenarios

5.1 Understanding the Game Flow

Here's what happens in a typical Wordle game to help you understand the function interactions:

1. **Game starts:** `initializeGame()` sets up everything
2. **Player types "A":** `handleKeyPress("A")` adds it to current guess
3. **Player types "B", "O", "U", "T":** Each call to `handleKeyPress()` builds the word
4. **Player presses ENTER:** `handleKeyPress("ENTER")` calls `submitGuess()`
5. **Word gets checked:** `submitGuess()` calls `checkLetter()` for each letter
6. **Colors revealed:** `processRowReveal()` handles any celebration effects
7. **Keyboard updated:** `updateKeyboardColors()` marks used letters
8. **Game continues or ends:** `updateGameState()` decides what's next

5.2 Duplicate Letter Examples

The trickiest part is handling duplicate letters correctly. Here are examples:

- **Target: "SPEED", Guess: "ERASE"**
  - E (pos 0): Not in position 0 of SPEED, but E exists → **Yellow**
  - R (pos 1): Not in SPEED → **Gray**
  - A (pos 2): Not in SPEED → **Gray**
  - S (pos 3): Not in position 3 of SPEED, but S exists → **Yellow**
  - E (pos 4): Matches position 4 of SPEED → **Green**
- **Target: "SPEED", Guess: "KEEPS"**
  - K (pos 0): Not in SPEED → **Gray**
  - E (pos 1): Matches position 1 of SPEED → **Green**
  - E (pos 2): Matches position 2 of SPEED → **Green**
  - P (pos 3): Matches position 3 of SPEED → **Green**
  - S (pos 4): Not in position 4, but S exists → **Yellow**

6 Testing & Debugging

6.1 Test Cases

Your implementation should handle these scenarios correctly:

Scenario	Input	Expected Behavior
Valid guess	"ABOUT"	Process guess, update colors
Invalid word	"XYZQW"	Show error, shake row
Duplicate letters	"SPEED" vs "ERASE"	Handle E's correctly
Win condition	Correct guess	Show celebration, modal
Lose condition	6 wrong guesses	Show game over modal
Keyboard input	Physical keys	Same as on-screen clicks

Table 3: Required Test Cases

6.2 Debugging Tools

The template includes debugging functions (remove before submission):

- `window.debug.revealWord()`: Show current target word
- `window.debug.autoSolve()`: Automatically solve current game
- `window.debug.gameState()`: Display current game state

7 Grading Rubric

Category	Points	Criteria
Core Game Logic	60	Functions work correctly, handle edge cases
Advanced Features	30	UI responsive, proper feedback
Code Quality	Bonus	Clean, readable, well-commented
Total	**90**	

Table 4: Point Distribution

7.1 Grade Scale

- **A (81-90)**: All functions implemented correctly with proper feedback
- **B (72-80)**: Core logic works, minor issues with advanced features
- **C (63-71)**: Basic game playable, some functions incomplete
- **D (54-62)**: Partial implementation, major functionality missing
- **F (0-53)**: Non-functional or minimal implementation

8 Submission Guidelines

8.1 What to Submit

1. **student-implementation.js**: Your completed JavaScript code
2. **README.md**: Brief description of your implementation approach

3. **Screenshots**: Show your working game (win and lose screens)

8.2 Before Submission

- Remove all debugging code and console.log statements
- Test all game scenarios thoroughly
- Ensure code is properly commented
- Verify game feedback works correctly
- Check that statistics persist between games

9 Common Pitfalls & Tips

Common Mistakes to Avoid

- **Duplicate Letter Logic**: The most common error is incorrectly handling words with repeated letters
- **Case Sensitivity**: Always convert to uppercase for comparisons
- **State Timing**: Update game state in the correct sequence
- **Input Validation**: Check for edge cases like empty inputs or game-over states
- **Keyboard Priority**: Don't downgrade key colors (green > yellow > gray)

Success Tips

- **Start Simple**: Implement basic functionality before adding animations
- **Test Frequently**: Use the debug functions to test edge cases
- **Read Documentation**: Study the provided utility functions carefully
- **Break Down Problems**: Tackle one function at a time
- **Use Console**: Log variables to understand program flow

10 Step-by-Step Implementation Guide

10.1 Recommended Implementation Order

Follow this sequence to build your Wordle game systematically:

1. **Start with `initializeGame()`**
   - This sets up everything and is called when the page loads
   - Test by opening the browser console and checking if variables are set
   - Use `window.debug.currentWord()` to verify a word was selected

2. **Implement basic `handleKeyPress()`**
   - Start with just letter keys - add them to `currentGuess`
   - Update the tile display so you can see letters appear
   - Add BACKSPACE functionality to remove letters
   - Test by typing letters and seeing them appear on the board
3. **Create simple `checkLetter()`**
   - Start with just the basic logic (ignore duplicates for now)
   - Test with simple words that don't have duplicate letters
   - Use `console.log()` to verify correct/present/absent results
4. **Build `submitGuess()`**
   - Update tile colors immediately for instant feedback
   - Test word validation with both valid and invalid words
   - Verify that colors appear correctly on tiles
5. **Add `updateGameState()`**
   - Handle win/lose detection
   - Test by intentionally winning and losing games
6. **Enhance with advanced features**
   - Add row completion effects with `processRowReveal()`
   - Implement keyboard color updates
   - Add input validation
   - Polish the end-game modal
7. **Handle duplicate letters in `checkLetter()`**
   - This is the most complex part - save it for last
   - Test with words like "SPEED", "LEVEL", "ALLEY"

10.2 Testing Your Progress

At each step, test your implementation:

- **Use the debug console:** `window.debug.revealWord()` shows the target
- **Try edge cases:** Empty inputs, invalid words, duplicate letters
- **Check the console:** Look for JavaScript errors or warnings
- **Test systematically:** Try to win, lose, and play multiple games

11 Extension Opportunities

For students seeking additional challenges:

- **Hard Mode**: Revealed hints must be used in subsequent guesses
- **Daily Words**: Implement a seed-based word selection
- **Share Results**: Generate shareable emoji grids
- **Accessibility**: Add screen reader support and keyboard navigation
- **Mobile Optimization**: Improve touch interactions

Good luck, and have fun building your Wordle clone!