JavaScript Events Quiz Game - Step-by-Step Implementation Guide

This guide will walk you through implementing the JavaScript Events Quiz Game one step at a time. This is a review activity to practice your understanding of JavaScript events.

Getting Started

1. First, open all the project files in your code editor:
   - `index.html` - Contains the pre-built HTML structure
   - `styles.css` - Contains the CSS styling
   - `quiz-starter.js` - This is where you'll be writing your code
2. Examine the HTML structure to understand the elements you'll be working with:
   - Welcome screen with name input
   - Quiz container with questions and options
   - Results screen for displaying score
3. Open the `quiz-starter.js` file and notice:
   - The quiz questions are already provided
   - Global variables are declared
   - DOM elements are selected
   - Function outlines are provided

Step 1: Initialize the Quiz with Window Events

1. Find the comment that says "Add window event listener to load the quiz when the page is ready"
2. Add the following code:

```javascript
// Initialize the quiz when the DOM is loaded
window.addEventListener('DOMContentLoaded', function() {
    console.log("DOM loaded - quiz initialized");

    // Set up total questions display
    totalQuestionsEl.textContent = quizQuestions.length;
    maxScoreEl.textContent = quizQuestions.length;

    // Show keyboard shortcuts guide (optional)
    shortcutsGuide.classList.remove('d-none');
});
```

Explanation:

- `DOMContentLoaded` event fires when the HTML document is completely loaded
- Inside the event handler, we initialize display elements
- The `classList.remove()` method makes an element visible by removing the 'd-none' Bootstrap class

Step 2: Handle Player Name Input with Form Events

1. Find the comment about adding form event listener for name submission
2. Add the following code:

```javascript
// Add form submission event listener
playerForm.addEventListener('submit', function(e) {
    e.preventDefault(); // Prevent form submission

    // Validate name
    if (validatePlayerName()) {
        playerName = playerNameInput.value.trim();
        startQuiz();
    }
});
```

3. Next, implement the blur event for validation:

```
// Add blur event listener to validate name
playerNameInput.addEventListener('blur', function() {
    validatePlayerName();
});

// Name validation function
function validatePlayerName() {
    const name = playerNameInput.value.trim();

    if (name.length < 3) {
        nameError.classList.remove('d-none');
        playerNameInput.classList.add('is-invalid');
        return false;
    } else {
        nameError.classList.add('d-none');
        playerNameInput.classList.remove('is-invalid');
        playerNameInput.classList.add('is-valid');
        return true;
    }
}
```

Explanation:

- The `submit` event fires when the form is submitted
- `e.preventDefault()` stops the form from submitting and refreshing the page
- The `blur` event fires when the input loses focus
- We validate that the name is at least 3 characters long
- Adding/removing Bootstrap classes provides visual feedback

Step 3: Implement Quiz Start Function

1. Find the `startQuiz()` function outline and fill it with:

```javascript
function startQuiz() {
    // Display player name
    playerNameDisplay.textContent = playerName;

    // Initialize quiz data
    currentQuestion = 0;
    score = 0;
    userAnswers = Array(quizQuestions.length).fill(null);

    // Show quiz container, hide welcome screen
    welcomeScreen.classList.add('d-none');
    quizContainer.classList.remove('d-none');
    quizContainer.classList.add('fade-in');

    // Show the first question
    showQuestion();

    // Start timer (optional)
    startTimer();
}
```

Explanation:

- We display the player's name in the quiz header
- Initialize variables to track progress and score
- `Array(length).fill(null)` creates an array to store user answers
- We update the display by toggling visibility of elements
- Call function to display the first question
- Optionally start a timer (we'll implement this later)

Step 4: Implement the Question Display Function

1. Find the `showQuestion()` function outline and complete it:

```javascript
function showQuestion() {
    // Get the current question data
    const questionData = quizQuestions[currentQuestion];

    // Set question text
    questionText.textContent = questionData.question;

    // Clear previous options
    optionsContainer.innerHTML = '';

    // Create and add options
    questionData.options.forEach((option, index) => {
        const optionElement = document.createElement('div');
        optionElement.className = 'option';
        optionElement.textContent = option;
        optionElement.dataset.index = index;

        // Check if this option was previously selected
        if (userAnswers[currentQuestion] === index) {
            optionElement.classList.add('selected');
        }

        // Add click event listener
        optionElement.addEventListener('click', function() {
            selectOption(index);
        });

        // Add mouseover and mouseout effects
        optionElement.addEventListener('mouseover', function() {
            this.style.boxShadow = '0 4px 8px rgba(0, 0, 0, 0.1)';
        });

        optionElement.addEventListener('mouseout', function() {
            this.style.boxShadow = 'none';
        });

        optionsContainer.appendChild(optionElement);
    });

    // Update the question counter
    currentQuestionEl.textContent = currentQuestion + 1;

    // Show/hide prev/next/submit buttons
    prevBtn.style.visibility = currentQuestion === 0 ? 'hidden' : 'visible';

    if (currentQuestion === quizQuestions.length - 1) {
        nextBtn.classList.add('d-none');
        submitBtn.classList.remove('d-none');
    } else {
        nextBtn.classList.remove('d-none');
        submitBtn.classList.add('d-none');
    }

    // Update progress bar
    updateProgress();
}
```

Explanation:

- We dynamically create option elements based on the current question
- Each option gets multiple event listeners:
    - `click` event to select the option
    - `mouseover` and `mouseout` events for visual effects
- We check if an option was previously selected and apply the 'selected' class
- Show/hide navigation buttons based on question position
- Update the progress display

Step 5: Implement Option Selection

1. Find the `selectOption()` function outline and complete it:

```
function selectOption(optionIndex) {
    // Update user answers
    userAnswers[currentQuestion] = optionIndex;

    // Update UI to show selected option
    const options = optionsContainer.querySelectorAll('.option');
    options.forEach((option, index) => {
        if (index === optionIndex) {
            option.classList.add('selected');
        } else {
            option.classList.remove('selected');
        }
    });
}
```

Explanation:

- Store the selected option index in the userAnswers array
- Get all option elements and update their appearance
- Add 'selected' class to the chosen option
- Remove 'selected' class from all other options

Step 6: Implement Question Navigation

1. Add event listeners for the navigation buttons:

```
// Handle next button click
nextBtn.addEventListener('click', nextQuestion);

function nextQuestion() {
    if (currentQuestion < quizQuestions.length - 1) {
        currentQuestion++;
        showQuestion();
    }
}

// Handle previous button click
prevBtn.addEventListener('click', prevQuestion);

function prevQuestion() {
    if (currentQuestion > 0) {
        currentQuestion--;
        showQuestion();
    }
}
```

2. Implement the progress bar update function:

```
function updateProgress() {
    const progress = ((currentQuestion + 1) / quizQuestions.length) * 100;
    progressFill.style.width = `${progress}%`;
}
```

Explanation:

- Add click event listeners to the navigation buttons
- The navigation functions update the currentQuestion index
- Check boundary conditions to prevent going beyond the first/last question
- Update the progress bar width based on current position

Step 7: Implement Quiz Submission

1. Add an event listener for the submit button and complete the submitQuiz function:

```
// Handle quiz submission
submitBtn.addEventListener('click', submitQuiz);

function submitQuiz() {
    // Calculate score
    score = 0;
    userAnswers.forEach((answer, index) => {
        if (answer === quizQuestions[index].answer) {
            score++;
        }
    });

    // Hide quiz, show results
    quizContainer.classList.add('d-none');
    resultsScreen.classList.remove('d-none');
    resultsScreen.classList.add('fade-in');

    // Display score
    scoreEl.textContent = score;

    // Display message based on score
    const percentage = (score / quizQuestions.length) * 100;

    if (percentage >= 80) {
        resultsMessage.textContent = "Excellent! You're a JavaScript events expert!";
        resultsMessage.className = "lead text-success";
    } else if (percentage >= 60) {
        resultsMessage.textContent = "Good job! You understand most JavaScript events.";
        resultsMessage.className = "lead text-primary";
    } else {
        resultsMessage.textContent = "You might need more practice with JavaScript events.";
        resultsMessage.className = "lead text-warning";
    }

    // Create answer review
    answerReview.innerHTML = '';
    quizQuestions.forEach((question, index) => {
        const isCorrect = userAnswers[index] === question.answer;

        const answerItem = document.createElement('div');
        answerItem.className = `answer-item ${isCorrect ? 'correct-answer' : 'incorrect-answer'

        answerItem.innerHTML = `
```

```
            <p><strong>Question ${index + 1}:</strong> ${question.question}</p>
            <p>Your answer: ${userAnswers[index] !== null ? question.options[userAnswers[index]]
            <p>Correct answer: ${question.options[question.answer]}</p>
        `;

        answerReview.appendChild(answerItem);
    });
}
```

Explanation:

- Compare user answers with correct answers to calculate score
- Toggle display between quiz and results screen
- Show different messages based on score percentage
- Create a detailed review of each question with correct/incorrect indicators

Step 8: Implement Restart Functionality

1. Add event listener for the restart button:

```
// Handle restart button
restartBtn.addEventListener('click', restartQuiz);

function restartQuiz() {
    // Reset all quiz data
    currentQuestion = 0;
    score = 0;
    userAnswers = [];

    // Return to welcome screen
    resultsScreen.classList.add('d-none');
    welcomeScreen.classList.remove('d-none');
    playerNameInput.value = '';
    playerNameInput.classList.remove('is-valid');
}
```

Explanation:

- Reset all quiz variables to their initial state
- Show the welcome screen again

- Clear the name input field

Step 9: Add Keyboard Navigation (Optional)

1. Add a keyboard event listener to enable keyboard navigation:

```
// Keyboard navigation
window.addEventListener('keydown', function(e) {
    // Only process keyboard events if we're in the quiz
    if (quizContainer.classList.contains('d-none')) return;

    switch(e.key) {
        case 'ArrowRight':
            if (currentQuestion < quizQuestions.length - 1) {
                nextQuestion();
            }
            break;
        case 'ArrowLeft':
            if (currentQuestion > 0) {
                prevQuestion();
            }
            break;
        case 'Enter':
            if (currentQuestion === quizQuestions.length - 1) {
                submitQuiz();
            }
            break;
        case '1':
        case '2':
        case '3':
        case '4':
            const optionIndex = parseInt(e.key) - 1;
            if (optionIndex >= 0 && optionIndex < quizQuestions[currentQuestion].options.length)
                selectOption(optionIndex);
            }
            break;
    }
});
```

Explanation:

- Listen for keyboard events on the entire window
- Only process events when the quiz is visible
- Left/Right arrows navigate between questions
- Number keys 1-4 select answer options
- Enter key submits the quiz on the last question

Step 10: Add Timer Functionality (Optional)

1. Implement the timer functions:

```
// Timer functionality
function startTimer() {
    timeLeft = 60;
    timeRemaining.textContent = timeLeft;

    timerInterval = setInterval(function() {
        timeLeft--;
        timeRemaining.textContent = timeLeft;

        // Warning when time is running low
        if (timeLeft <= 10) {
            timerDisplay.classList.add('timer-low');
        }

        // Time's up
        if (timeLeft <= 0) {
            clearInterval(timerInterval);
            submitQuiz();
        }
    }, 1000);
}
```

2. Make sure to stop the timer when the quiz is submitted:

```
// Add this at the beginning of the submitQuiz function
function submitQuiz() {
    // Stop the timer
    clearInterval(timerInterval);

    // Rest of the function...
}
```

Explanation:

- Start a timer that counts down from 60 seconds
- Update the display every second using setInterval
- Add visual warning when time is running low
- Automatically submit when time reaches zero
- Clear the interval to stop the timer when quiz is submitted

Testing Your Implementation

After implementing each step:

1. Save your files
2. Open index.html in your browser
3. Test the functionality you just added
4. Check the console for any errors (F12 to open Developer Tools)
5. Fix any issues before moving to the next step

Final Steps

1. Test your complete quiz:
    - Try submitting without a name
    - Try selecting different answers
    - Navigate with buttons and keyboard
    - Check if the score is calculated correctly
    - Test the restart functionality
2. Debug any remaining issues
3. Review your code and add comments where needed
4. Consider adding any extra features you'd like!

Congratulations! You've successfully implemented a complete quiz application using JavaScript events!