1. Implementation Plan Document

1.1 Introduction

This document outlines the plan for developing a Tetris game using JavaScript, HTML5 Canvas, and CSS. The game will include standard Tetris features, such as random piece generation, movement, rotation, line clearing, scoring, and a game over state. Additional features will include a "next piece" preview, a start/pause button, a game over overlay, and clear instructions.

1.2 Goals

- Develop a fully functional and playable Tetris game in the browser.
- Implement all core Tetris game mechanics.
- Enhance the game with user-friendly features.
- Create a visually appealing and responsive game interface.
- Document the development process.

1.3 Features

- **Core Features:**
  - 12x20 game board.
  - Seven Tetromino shapes (I, J, L, O, S, T, Z).
  - Random piece generation.
  - Left/right/down movement.
  - Clockwise and counter-clockwise rotation.
  - Collision detection.
  - Board merging.
  - Line clearing.
  - Scoring.
  - Game over condition.
- **Added Features:**
  - Next piece preview.
  - Start/pause/restart button.
  - Game over overlay.
  - Basic scoring system.
  - Clear instructions and rules.

1.4 Technologies

- HTML5 Canvas for rendering the game board and pieces.
- JavaScript for game logic and interactivity.
- CSS for styling the game interface.

1.5 Development Steps

1. **Project Setup:**
   - Create index.html, style.css, and tetris.js files.
   - Set up the basic HTML structure with canvases, buttons, and score display.
   - Link CSS and JavaScript files.
   - Initialize the game canvas and context in tetris.js.
   - Set up basic CSS styling for the game container and elements.
2. **Game Board and Drawing Functions:**
   - Implement the createBoard() function to represent the game board as a 2D array.
   - Implement drawSquare() and drawMatrix() functions to render blocks and tetrominoes on the canvas.
   - Implement drawNextPiece() to display the upcoming piece.
3. **Tetromino Logic:**
   - Define the Tetromino shapes and colors.
   - Implement generateNewPiece() to randomly select a Tetromino.
   - Implement merge() to add a Tetromino to the board.
   - Implement rotate() and playerRotate() functions for piece rotation, including wall kick logic.
4. **Game Mechanics:**
   - Implement collide() function for collision detection.
   - Implement playerMove() and playerDrop() functions for Tetromino movement.
   - Implement boardSweep() function for detecting and clearing completed lines.
   - Implement scoring logic within boardSweep().
   - Implement playerReset() function and game over detection.
5. **Game Loop and Input Handling:**
   - Implement the update() and draw() functions for the game loop using requestAnimationFrame().
   - Implement keyboard event listeners for player controls.
   - Implement button event listener for start/pause/restart functionality.
6. **User Interface Enhancements:**
   - Implement the "Next Piece" preview.
   - Implement the game over overlay.
   - Ensure the score is displayed and updated correctly.
   - Add clear instructions and rules to the UI.
7. **Testing and Refinement:**
   - Thoroughly test all game features.
   - Debug and fix any issues.

    - Refine the game logic and UI for optimal performance and user experience.
8. **Documentation:**
    - Write a comprehensive README file.
    - Create implementation plan.
    - Create reflection document.

1.7 Resources

- HTML5 Canvas API Documentation
- JavaScript Documentation
- Online tutorials and resources for Tetris game development

2. Reflection Document

2.1 Introduction

This document reflects on the development process of the Tetris game, including successes, challenges, lessons learned, and potential improvements.

2.2 Project Goals Review

* Develop a fully functional and playable Tetris game in the browser: Achieved. The game is playable with all core mechanics implemented.
* Implement all core Tetris game mechanics: Achieved. All core features (movement, rotation, collision, scoring, game over) are implemented.
* Enhance the game with user-friendly features: Achieved. Features like the next piece preview, start/pause/restart button, and clear instructions improve the user experience.
* Create a visually appealing and responsive game interface: Partially Achieved. The game has a functional UI, but further styling could enhance visual appeal.
* Document the development process: Achieved. This reflection document and the implementation plan serve as documentation.

2.3 Successes

- Successfully implemented all core Tetris game mechanics.
- Added user-friendly features that enhance the gameplay.
- Developed a functional and responsive game within the planned timeline.
- Gained experience with HTML5 Canvas, JavaScript game development, and game logic.
- The game is playable and relatively bug-free.

2.4 Challenges and Solutions

- **Game Over Condition:** Detecting the precise moment of game over was tricky.
    - **Solution:** Checked for collision upon new piece generation to determine if the game is over.
- **Next Piece Preview:** Displaying the next piece required careful calculation.
    - **Solution:** Created a separate canvas and centered the piece on it.

2.5 Lessons Learned

- Game development requires careful planning and attention to detail, especially with collision detection and movement.
- Thorough testing is crucial to identify and fix bugs early in the development process.
- Using a version control system (like Git) is essential for managing code changes and collaborating (if applicable).
- Clear and concise code documentation improves maintainability and understanding.

2.6 Areas for Improvement

- **Visual Enhancements:** Improve the game's visual appearance with better styling, animations.
- **Advanced Scoring:** Implement a more complex scoring system with levels.
- **User Interface:** Further refine the UI, perhaps with a more polished start screen, pause menu, and game over screen.
- **Code Refactoring:** Refactor the code for better organization, readability, and maintainability.
- **Error Handling:** Implement more robust error handling.
- **Modularity:** Break down the code into more modular components.
- **Performance:** Optimize the game loop and rendering for smoother performance.