SOFTWARE PROJECT FINAL REPORT

Rebeca Oliveira de Souza and Colin McClenaghan

December 5, 2024

Table of Contents

List of Tables

1. Introduction

1.1. Purpose and Scope

The purpose of this project was to develop a single-player Tetris game in C# as part of the CIS434 course. The game adheres to standard Tetris mechanics, providing smooth performance and a user-friendly interface.

1.2. Product Overview (including capabilities, scenarios for using the product, etc.)

The Tetris game challenges players to clear rows by strategically arranging falling tetrominoes. The game includes features such as:

- Randomized tetromino generation (7-bag system).
- Tetromino holding and preview queue.
- Scoring and level progression.
- Core mechanics like line clearing, rotation, and collision detection.

1.3. Structure of the Document

This report documents the development process, including project management, requirements, architecture, design, testing, and conclusions.

1.4. Terms, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **SDS**: Software Design Specification
- **RM3**: Risk Mitigation, Monitoring, and Management
- **SRS**: Super Rotation System for tetromino handling.
- **7-bag**: Randomization system ensuring uniform tetromino distribution.

2. Project Management Plan

2.1. Project Organization

This project was worked on by two team members, with the tasks being divided in an unique manner to play to each other's strengths

- Rebeca: Most of the documentation, and some implementation of features.
- Colin: Most of the code, game logic and graphics rendering.

2.2. Lifecycle Model Used

An iterative development model was adopted, ensuring that critical features were developed and tested incrementally.

2.3. Risk Analysis

Based on our initial predictions what affected us the most was scheduling conflicts and getting everything done on time, and using C# for the game.

Below is the initial table of risks with all that we thought could impact us.

Risk	Probability	Impact	RM3 Pointer
Schedule conflicts	High	Low-Medium	Regular Progress Check-ins
Delayed Delivery	Low	High	Time Management
Team member gets sick	Medium	High	Refer to backup plan in RM3
Performance issues (slow rendering)	Medium	Medium	See testing protocol in RM3
Unclear requirements (scope creep)	Low	High	Refer to requirements management
Unfamiliarity with C#	Medium	Medium	Refer to learning resources

Table 2.1 - Risk Management Summary

2.4. Hardware and Software Resource Requirements

Hardware: Standard Windows 10 PC with 4GB RAM.

Software: Visual Studio 2022, .NET Core.

2.5. Deliverables and schedule

Initial prototype: November 14, 2024

Final submission (user guide, developer guide and project report): December 5, 2024

3. Requirement Specifications

3.1. Stakeholders for the system

Primary stakeholders include casual gamers and the CIS434 course instructor.

3.2. Use cases

1. **Start a new game**: The player starts a fresh game session from the main menu.
2. **Control tetrominoes**: The player moves, rotates, and drops blocks using keyboard inputs.
3. **Score calculation**: Rows are cleared, and scores are updated based on player performance.
4. **Game over**: The game ends when blocks reach the top of the screen, and the player is presented with a score and the option to restart.
5. **Error Handling:** If a player presses an invalid key or an unrecognized input, the game should ignore the input and maintain a smooth experience without crashing.

3.3. Rationale for your use case model

The selected use cases were chosen to encompass the core functionality and user interactions essential for a Tetris game. Each use case aligns with the game's objectives and ensures the system operates effectively within the project's scope. The rationale for these use cases is as follows:

- **Start a New Game**: This use case represents the entry point to the game, ensuring that players can easily begin a session. It is a critical interaction to provide a seamless start for gameplay.
- **Control Tetrominoes**: This use case addresses the primary interaction between the player and the game. The ability to move, rotate, and drop tetrominoes is central to the gameplay experience, making it the most important use case.
- **Score Calculation**: Scoring is a key motivator for players and a measure of their success. By incorporating this use case, the game provides immediate feedback and progression, encouraging continued play.
- **Game Over**: This use case defines the conclusion of a game session, presenting the player with their performance outcome and an option to restart. It ensures the game has a clear and logical end condition, enhancing user experience.

- **Error Handling**: This use case ensures robustness by ignoring invalid inputs, which prevents crashes and maintains a smooth gameplay experience. It contributes to the system's usability and reliability, crucial for a polished product.

By prioritizing these use cases, the model ensures that the game's fundamental mechanics and user interactions are fully realized, providing a satisfying and error-free experience for players.

3.4. Non-functional requirements

- **Performance**: The game must run at 60 FPS with minimal lag.
- **Usability**: Simple keyboard controls and clear visuals.

4. Architecture

4.1. Architectural style(s) used

A modular architecture was implemented to separate game logic, rendering, and input handling. This approach ensures clear boundaries between components, facilitating development, testing, and debugging.

4.2. Architectural model (includes components and their interactions)

- **Game Logic Module**: Handles collision detection, scoring, and line clearing.
- **Renderer Module**: Renders tetrominoes, gameboard, and UI elements.
- **Input Module**: Processes keyboard inputs for gameplay.

4.3. Technology, software, and hardware used

- **Language**: C#
- **Framework**: .NET Core

4.4. Rationale for your architectural style and model

The modular design ensures maintainability, scalability and testability.

5. Design

5.1. User Interface design

The game interface includes:

- A **grid-based game board** for tetromino gameplay.
- A **preview queue** for the next pieces.
- A **hold box** for reserved tetrominoes.
- **Score and level indicators** displayed prominently on the screen.
  The layout is minimalist, ensuring user focus remains on gameplay.

5.2. Components design (static and dynamic models of each component)

Static Models:

- Tetromino Class: Represents individual game pieces with properties for shape, rotation state, and position.
- Board Class: A 2D array representing the grid where tetrominoes fall and settle.
- ScoreManager Class: Tracks the current score and level progression.

Dynamic Models:

- Game Loop: Continuously updates the game state, checks for collisions, clears lines, and increases difficulty as needed.
- Input Handling: Captures keyboard events and triggers corresponding game logic updates.
- Rendering: Redraws the board and tetrominoes at a consistent frame rate.

5.3. Database design

No database was used or needed

5.4. Rationale for your detailed design models

The detailed design aligns with the modular architecture, ensuring each component's responsibility is clearly defined. This separation simplifies debugging and allows for incremental feature development.

5.5. Traceability from requirements to detailed design models

- **Requirement**: Tetromino control → **Component**: Input Module.
- **Requirement**: Line clearing → **Component**: Game Logic Module.
- **Requirement**: Score tracking → **Component**: ScoreManager Class.
- **Requirement**: Gameboard rendering → **Component**: Renderer Module.

6. Test Management

6.1. A complete list of system test cases

- **Corrupted Settings File**: Ensures default settings overwrite invalid configuration files.
- **Corrupted Save File**: Validates that invalid save files are ignored during loading.
- **Window Resizing**: Confirms that UI elements adjust correctly when resizing the window to extreme dimensions.
- **Game Unfocused Test**: Verifies the game pauses automatically when the application loses focus.
- **Game Logic Testing**: Checks the behavior of core mechanics like tetromino rotation, line clearing, and gravity across edge cases.

6.2. Traceability of test cases to use cases

- **Use Case**: Start a new game → **Test Case**: Proper initialization of settings and game state.
- **Use Case**: Tetromino movement and rotation → **Test Case**: Input handling and collision checks.
- **Use Case**: Line clearing → **Test Case**: Scoring accuracy and line removal.

6.3. Techniques used for test case generation

- **Incremental Testing**: Critical components like game logic and input handling were tested as they were implemented.
- **Edge Case Validation**: Preset data and scenarios were used to ensure stability under extreme conditions (e.g., rapidly dropping pieces).
- **Visual Testing**: Observations of the game interface and interactions helped detect UI-related issues.

6.4. Test results and assessments (how good are your test cases? How good is your software?)

- **Critical Features**: Passed initial and edge-case testing. These include tetromino movement, rotation, line clearing, and game-over scenarios.
- **Non-Critical Features**: Pause functionality and resizing logic passed after minor adjustments.
- **Software Quality**: High reliability was achieved through thorough testing during and after implementation.

6.5. Defects reports

7. Conclusions

7.1. Outcomes of the project (are all goals achieved?)

The project successfully implemented the core mechanics of Tetris, achieving smooth performance and a polished interface. All primary objectives were met, like properly writing the game logic

7.2. Lessons learned

The Largest issue when working on the project was time constraints.

Several features were cut due to time limitations. It is a good lesson to take away to manage the time spent better.

7.3. Future development

future development would likely involve adding the features that were left out of this version as well as a multiplayer mode.

Potential improvements:

- Adding multiplayer functionality.
- Enhancing graphics with animations and visual effects.
- Implementing online leaderboards for high scores.
- Adding a menu

References

Tetris guidelines the project is based on:

https://github.com/Broderick-Westrope/tetrigo/blob/main/docs/2009-Tetris-Design-Guideline.pdf