CSE 211
Data Structures
TERM PROJECT FALL 2023

Tetris Game

1 Overview

This document outlines the implementation strategy for a custom Tetris game in C++. The game is designed with several advanced features like a shrinking board, combo system, piece preview and hold, and customizable themes. It uses a frame-by-frame rendering approach in the terminal, providing a classic yet enriched gaming experience.

2 Class Structure

2.1 Game Management

2.1.1 Game:

Central class managing the game loop, state, and progression. It orchestrates the interaction between different components like the board, pieces, and user inputs.

2.1.2 Board:

Represents the Tetris playfield. It manages the grid where pieces are placed and handles line completion, board shrinking, and special features like portal cells.

2.2 Pieces and Variants

2.2.1 Piece:

The Piece class uses an enumeration to define standard piece types and includes support for additional custom types.

Users can add up to three custom pieces to play with their shape matrices.

2.3 Game Features

2.3.1 Piece Preview:

Manages the display of the next upcoming piece, allowing players to strategize their placement.

2.3.2 Piece Hold:

Allows players to hold a piece for later use, adding strategic depth to the game.

2.3.3 Theme Manager and Theme:

Manages customizable themes for the game, allowing players to personalize visuals like colors and styles.

2.3.4 Audio Manager

Handles background music and sound effects, enhancing the game's audio experience. You can use any music without copyright.

3 Key Features and Implementation Strategy

3.1 Frame-by-Frame Rendering

- The game is rendered frame-by-frame in a terminal.
- Each frame represents a snapshot of the game state, updated at regular intervals.

- Implement a method in the Game class to refresh the display each frame, clearing the terminal and re-drawing the entire game state.

3.2 Shrinking Board

- Implement dynamic grid resizing within the **Board** class.
- Trigger board shrinking based on level or specific conditions.
- Adjust piece placement logic to accommodate the changing board size.

3.3 Combo System

- Monitor and update combos based on consecutive line clears.
- Enhance scoring logic in the Game class to incorporate combo multipliers.

3.4 Piece Preview and Hold

- Use Piece Preview to display the next piece.
- Implement Piece Hold logic to allow players to save and swap a piece once per turn.
- Update game loop in Game to handle piece preview and hold actions.

3.5 Customizable Themes

- Utilize Theme Manager to switch between different visual themes.
- Allow players to select and apply themes, affecting colors and styles in the game render.

3.6 Environmental Effects

- Dynamic Changes: Implement changes in the game environment, such as varying gravity, speed alternations, or random disruptions.
- Trigger Mechanism: Design a system within the Game class to trigger environmental effects based on game progression or random events.
- Effect Management: Manage the duration and intensity of effects, ensuring they provide challenge without hindering playability.

3.7 Portal Mechanics

- Portal Placement: Integrate Portal cells withing the Board grid, determining their positions and behaviors.
- Teleportation Logic: Implement logic in the Board class to handle the teleportation of pieces when they interact with portal cells.
- Visual Indicators: Provide clear visual cues for portal locations and activations, ensuring players can plan their moves accordingly.

3.8 Limited Rotations

- Rotation Counter: Integrate a rotation limit for each piece, managed by the Piece class and its derivatives.
- User Feedback: Provide immediate feedback to the player when a rotation is not possible

3.9 Terminal Interactivity

- Capture user input in real-time for piece movement and rotation (“q” and “e” can be used for rotations, up and down arrows can be used for movement.
- Implement efficient input handling to ensure responsive gameplay.
- Implement menu:
  - Resume (If game is already started)
  - New Game
    - Difficulty settings etc.
    - Load Custom Pieces
- Score Board (User should write his uuid)
- Theme Selection
- Quit

4 Pieces

All pieces should be derived from matrix. User can give and create their own Pieces with their shape matrix. Matrix shape can be anything (3x3, 2x2, 1x3, 2x3 etc.)

For example, Plus-Shaped Piece (Cross) shape matrix:

0	1	0
1	1	1
0	1	0

Here `1` represents a filled block and `0` an empty space.

Plus Sign	0,1,0;1,1,1;0,1,0
Small Square	1,1;1,1
Corner Piece	1,0;1,1
Tall Tower	1;1;1
Zigzag	0,1,1;1,1,0
U Shape	1,0,1;1,1,1
Step Shape	1,0,0;1,1,0;0,1,0
Arrow Pointing Up	0,1,0;1,1,1;1,0,1
Double Zigzag	1,0,0;1,1,1;0,0,1
Pyramid	0,1,0;1,1,1

5 Development Approach

- Follow Object-Oriented Programming principles for clarity and modularity.
- Test each component individually before integrating into the main game loop.
- Prioritize efficient rendering for a smooth terminal-based experience.
  - When the frame changes, you should delete the old frame's output from the terminal.
- Implement robust error handling and edge case management.
- You can find how to rotate pieces here.
- You can not use any prebuilt data structure classes, like:
  - Vector,
  - Stack,
  - Queue,
  - Graph,
- You need to implement your Shape (Might be inside Piece class) and Grid (Might be inside Board class) structures.

6 Report:

A comprehensive one-page documentation of your project should be provided. Check the proper report example on web.

- Introduction: Start with a brief introduction to the problem you were trying to solve and the objectives of your code.
- Methodology: Describe the approach you took to solve the problem and the algorithms you used. Explain any new concepts or techniques you learned while working on the project.
- Implementation: Detail the steps you took to implement the solution in code. Discuss any trade-offs you made and the reasons behind them. Explain any key decisions you made while writing the code, such as data structures or libraries you used.
- Results: Present the results of your code.
- Conclusion: Summarize the key points of your report and highlight the significance of your findings. Discuss any limitations of your code and suggest areas for future improvement.

7 Submission

- You should compress (Preferred extension is zip.) all your files and submit a single file named with your name and student ID.
- The zip file name should be following format: <**STUDENT_NAME**><**STUDENT_SURNAME**><**STUDENT_ID**>.zip example batuhanEdgüer2023000000.zip
  Just a reminder, your name and surname are not "Batuhan Edgüer".
- Do your own work to stay away from punishment. Good luck!
- You can ask all your questions via email (Click the hyperlink to send an instant email.), or you can visit me at the faculty.
- If your code isn't compiled on my computer, you will get zero credit for your assignment.
- Commented codes will be ignored.