CSE 211: Tetris Game Project

This project aimed to develop a Tetris game in C++. The primary objective was to create an interactive and functional implementation of the Tetris game, allowing users to experience the Tetris gameplay.

The project involved employing object-oriented programming principles to model game entities such as pieces and the game board. Algorithms were crucial for implementing piece movement, collision detection, line clearing, and score calculation, forming the core gameplay mechanics. Additionally, the project introduced concepts like game loops, input handling, and timing for piece drops. Also it was effective in learning new libraries ,their implementations and using threads in C++. This project could serve as a small entry point into game development.

As part of the project, there is an implementation of "Piece" and "Board" classes to represent pieces and the game board using matrices. Piece rotation, collision detection, and placement are managed through matrix manipulation. Additionally, there is a "Game" class used for the basic game loop, handling game states, updating the board, rendering the interface, and collecting user inputs. A console-based interface was chosen for rendering the game, sacrificing visual appeal for simplicity. The decision to use console I/O may limit rich graphical interactions. The game also features scoring and combo systems, which can be observed through console outputs. Libraries used:

-iostream: Standard print and console inputs in C++.

-conio.h: Basic I/O process. Also used for clrscr() to clearing screen, getch() to input handling.

-thread: Used for optimizng rendering, input handling and screen updating.

-chrono: Used for clock

-array: Used for matrix manipulation

-algorithm: Used for piece rotation.

Project succesfully created a basic Tetris game. While functional, limitations exist in terms of user interface and graphical representation. Future improvements might include transitioning to a graphical interface, adding sound effects, level system and random events.