Object Oriented Programming
Final Project–Tetris

Po-Han Liao
EE15
111511146

Kai-Hsun Chen
EE15
111511012

***Abstract*—This report is used to tell more detail and what we can do more to make the project better. We'll talk about introduction, code, expectation, and reference. In the end, we'll give a github link to let you download our code to play the game.**

I. Introduction

Tetris is a timeless and addictive puzzle game that originated in the Soviet Union in the 1980s.It challenges players to manipulate falling geometric shapes called "tetrominos" to create complete horizontal lines without any gaps. As the game progresses, the tetrominos fall faster, requiring quick thinking and reflexes. With its simple yet engaging gameplay, Tetris has become an iconic and widely recognized game that has been ported to numerous platforms and remains popular to this day. In our final project, we want to remake it in pygame by ourselves.

II. Code

The provided code is written in Python and utilizes the Pygame library to create an interactive Tetris game. Let's dive deeper into its functionality.

A. Game Loop

The game loop is the core component that runs continuously until the game ends or the player chooses to exit. It handles game updates, user input, and graphics rendering, ensuring smooth gameplay.

B. User Interaction

The code handles keyboard events using pygame.event.get(). It listens for key presses that allow the player to move the tetromino left or right, rotate it, or move it down. These actions are implemented through the **"Tetromino"** and **"GameBoard"** classes.

C. Game State Updates

The game state is continuously updated within the game loop. The code checks for collision between the tetromino and the game board, moves the tetromino down at regular intervals, and adds it to the game board once it reaches the bottom or collides with existing blocks.

D. Clearing Completed Rows

When a row is completely filled with blocks, the code clears that row from the game board and add a empty row at the top, the remaining blocks above the cleared row are shifted down to fill the gap.

E. Scoring

The player's score is determined by the number of completed rows. Each time a row is cleared, the player earns 1 point that are added to the score. The current score is displayed on terminator.

F. Victory Condition

If the player achieves a specific score threshold, which is 10 rows, indicating a win, the code reacts by stopping the background music, playing a victory sound effect, and displaying a "Congratulations!" message on terminator.

G. Game Over

When the player reaches the top of the game board, indicating a game over, the code responds by stopping the background music, playing a sound effect, and displaying a "You lose" message on terminator.

H. Graphics Rendering

The Pygame library's surface and drawing functions are used to render the game graphics. The code employs these functions to draw the game board, tetrominoes, grid lines, score text, and buttons. The **"pygame.display.update()"** function is called to refresh the display and reflect any changes made.

I. Audio Effects

Audio effects enhance the gameplay experience. The code uses Pygame's mixer module to load music files with **"pygame.mixer.Sound"** and plays them at appropriate times. For instance, background music during gameplay, sound effects when completing a row, and end-of-game audio cues.

J. Button Functionality

Buttons are implemented through a Button class. The code detects mouse clicks within the game loop and triggers specific actions based on the clicked button. For example, clicking the ”play” button starts the game, the ”rules” button displays the rules, and the ”exit” button in the rules screen returns to the initial screen.

III. Further Expectation

After live demo on 6/12, we found that others' projects have many good things that we can learn from. In conclusion, we find 4 things we can do to improve our project below.

A. Auto Game Play

Auto game plays could be a part of PVE in the future, and they may be implemented by calculating the weighted score of every available position. If a tetromino is placed in a certain position that can clear a row, the weighted score should be increased. On the other hand, if placing a tetromino creates a closed area that cannot be filled by other blocks, the weighted score should be decreased. Currently, the weights can only be determined based on the designer's gaming experience. For instance, canceling a line will give 5 points, and every empty block enclosed will deduct 1 point. However, with advancements in AI training, we may be able to discover the best strategy for weighting scores.

B. Different Game Setting

We can create more customized options for players, such as different board sizes and drop speeds, among others. This can be easily implemented thanks to the OOP coding structure by changing constants into input variables in the corresponding initialization function. Otherwise, we would need to change every input of every function to obtain a new variable.

C. Score Leaderboard

Adding a score leaderboard to record the time each player spent to finish ten lines will enhance the gameplay experience and generate greater enthusiasm among the players.

D. Multiplayer Mode

Let two players play on the same keyboard, and every line cleared will be added to the bottom of the other player's game board. The technical issue is how to allocate the keyboard. Modifying the code will be easy, thanks to the OOP structure once again. We just need to generate two objects with the same class but different variables and add the function that defines how the two objects interact within the class. Hence, we won't have to rewrite a bunch of duplicated functions in our code.

IV. References

- ChatGPT: debug and give a cleaner code.
- Youtube: the back ground music of the game, and all of the effect sound (coin, win, lose).
- Google: the ”TETRIS” picture.

V. Github link

https://github.com/kaihsun1014/group1finalproject/tree/main