Assignment 3: Tetris Rush

Intro to Computer Science (CS-UH 1001) - Spring 2022

1 Code of Conduct

All assignments are graded, meaning we expect you to adhere to the academic integrity standards of NYU Abu Dhabi. To avoid any confusion regarding this, we will briefly state what is and isn't allowed when working on an assignment.

1. Any document and program code that you submit must be fully written by yourself.
2. You can discuss your work with fellow students, as long as these discussions are restricted to general solution techniques. In other words, these discussions should not be about concrete code you are writing, nor about specific results you wish to submit.
3. When discussing an assignment with others, this should never lead to you possessing the complete or partial solution of others, regardless of whether the solution is in paper or digital form, and independent of who made the solution.
4. You are not allowed to possess solutions by someone from a different year or section, by someone from another university, or code from the Internet, etc.
5. There is never a valid reason to share your code with fellow students.
6. There is no valid reason to publish your code online in any form.
7. Every student is responsible for the work they submit. If there is any doubt during the grading about whether a student created the assignment themselves (e.g. if the solution matches that of others), we reserve the option to let the student explain why this is the case. In case doubts remain, or we decide to directly escalate the issue, the suspected violations will be reported to the academic administration according to the policies of NYU Abu Dhabi. More details can be found at: https://students.nyuad.nyu.edu/academics/registration/academic-policies/academic-integrity/

2 Introduction

The game described in this document is inspired by the famous game Tetris which was published in 1984 by Alexey Pajitnov. Tetris is a tile-matching puzzle video game which is primarily composed of a board in which pieces of different geometric forms descend from the top of the board. During the descent, the pieces can be freely moved and rotated until they touch the bottom of the board or land on a piece that had been placed before. The objective of Tetris is to create as many horizontal lines of blocks as possible to prevent the board from filling up. When a row is completed, all blocks of the row disappear and the blocks placed above are

moved one row down. Every completion of a row earns points and the goal is to earn as many points as possible before the board is entirely filled and the game is lost.

The concept of Tetris Rush follows a similar approach but is simplified in multiple aspects as follows:

1. Tetris Rush is an interactive game for one player, where randomly generated colored and single-squared blocks descend from the top of the board with a certain speed.
2. The blocks can be freely moved horizontally while they are descending. However, the speed of the falling blocks increases with every new block that is generated, leaving less time for the player to think about the placement as the game progresses.
3. The objective of Tetris Rush is to create groups of 4 consecutive and vertically aligned blocks with the same color to earn points and prevent the board from filling up.
4. When a group of 4 vertically aligned blocks with the same color is completed, all 4 blocks that form the group are removed from the board and the speed of falling blocks is reset to its initial value.
5. If the player cannot make the blocks disappear quickly enough, the board will start to fill up and when the blocks in all columns reach the top of the board and prevent the arrival of additional blocks, the game ends.

The objective of this assignment is to implement the game in Python using the Processing IDE. This will give you practice in working with Python, GUIs and Processing. Additionally, it is a good exercise to understand how to handle mouse and keyboard events, object-oriented programming and how to use the documentation of Processing which can be found here: https://processing.org/reference/

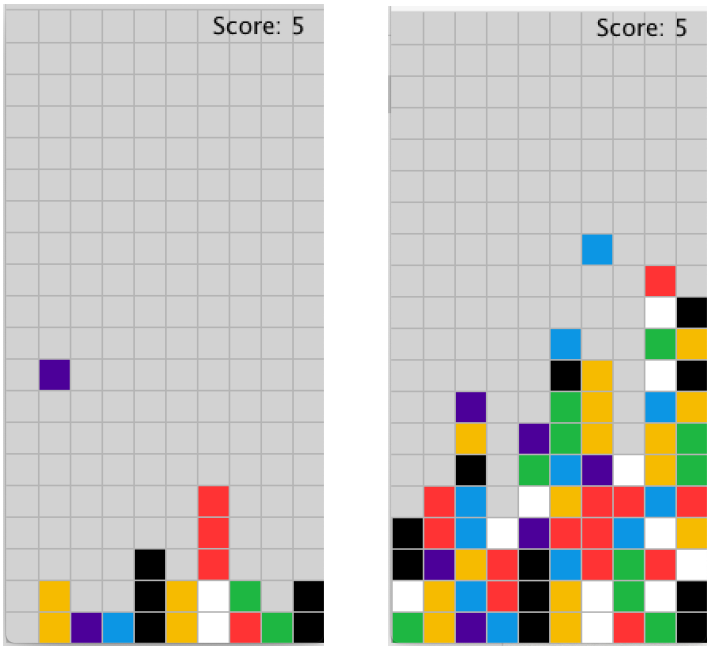

Figure 1: Screenshots of the Processing implementation

3 Implementation

The following implementation guidelines should be considered when implementing the game.

Initial setup:
The setup() function of Processing is used to do the initial setup of the game, e.g. board size, background, etc.. Initaly, the board should be empty and not filled with any blocks. The draw() function is used for game display and in order to slow down the speed of the game, the built-in variable frameCount should be used. Use the following code inside the draw() function to dynamically change the speed of the game (and hence the speed of the falling blocks) during runtime. The attribute game.speed is an attribute of the game class and should be initialized to 0 when the game starts:

```
def draw():
    #slow down the game by not calling the display() method every
frame
   if frameCount%(max(1, int(8 - game.speed)))==0 or frameCount==1:
           background(210)
           #this calls the display method of the game class
           game.display()
```

The attribute game.speed should be incremented by 0.25 for every block that is generated and reset to 0 when 4 consecutive blocks of the same color are removed from the board.

Board and board dimensions: The background color of the board should be 210 (grayscale, use background(210)) and should have grid lines displayed with color 180 (use stroke(180)). The board resolution should be 200x400 pixels and should be adjustable if desired, i.e. by global variables specifying the number of rows and columns. Each cell of the board has a size of 20x20 pixels, which results in 20 rows and 10 columns for the above mentioned board dimensions.

Python classes: The implementation should consist of at least 2 Python classes:

1. A class reflecting a block
2. A class reflecting the overall game

Each class should contain all necessary attributes and methods that are required to reflect the properties of the objects.

Blocks: A block has a size of 20x20 pixels and a color (see color scheme below). A new block is instantiated after a block has landed at the bottom of the board or on a block that had been placed before it in the same column. The color of a block should be randomly chosen when instantiated and the following color codes should be used:

Red: 255, 51, 52
Blue: 12, 150, 228
Green: 30, 183, 66
Yellow: 246, 187, 0
Purple: 76, 0, 153
White: 255, 255, 255
Black: 0, 0, 0

Whenever a falling block is placed at the bottom of the board or lands on a block that had been placed before it, a new block must be generated at the top of the board at a random location, i.e. at row 0 and a random column. No more than 1 falling block can exist at a time.
After a block has been placed on another block, it must be checked if 4 vertically adjacent blocks of the same color exist in the same column. If so, the 4 blocks are removed from the board and the space is cleared to make room for future blocks to be placed. Also, the attribute game.speed is reset to 0 to slow down the game to its initial speed.

Block movement: Every block falls with a fixed speed from the top to the bottom of the board, i.e. 1 row for every call of the display() method of the block class (which is called from within the display() method of the game class). As the frequency of how often the display() method of the game class is called is controlled by the draw() function of Processing (see Initial setup), the attribute game.speed controls how often a block is moved by 1 row and hence makes the block look fall faster or slower depending on the value of game.speed.
While a block is falling, the block can be freely moved to the right or left using the 2 arrow keys of the keyboard. The arrow keys are represented by the built-in keyCode variables LEFT and RIGHT of Processing.
(Check the documentation for more details: https://processing.org/reference/keyCode.html)
Every key press moves the block 1 position (column) to the right or left. The blocks can not be moved outside the board dimensions and must not overlap with blocks that are already placed on the board, i.e. two blocks can never be displayed on the same cell of the board.
Note: Since the display() method of the game class is not called every frame (see Initial setup), a key-pressed event should be transferred to the game class by calling a method that moves the block either to the right or left.
Score: The game should display a score in the upper right corner with a font size of 15. Use the following built-in functions to display a string:

```
textSize(15)
text(STRING)
```

The score starts with 0 at the beginning of the game and is incremented by 1 if 4 vertically adjacent blocks of the same color are removed from the board.

Game over: When the entire board is filled with blocks, the game ends and "Game over" together with the final score should be displayed on the screen. Upon mouse click anywhere on

the screen, the game starts from the beginning. Please note: As the column for generating new blocks is randomly chosen everytime a new block is generated, the game only ends if all columns are filled.

4 Grading

Description	**Score (/15)**
Following the implementation specs (colors, dimensions, background, etc)	1
Display the board including gridlines	1
Instantiation of new blocks ● random location at top of board ● randomly generated colors ● stop descend after a block has landed on either another block or the bottom of the board	3
Horizontal block movement ● right and left keyboard events ● not outside board dimensions ● no overlapping with already placed blocks	3
Increase in the speed of the falling blocks with every new block generated	1
When a group of 4 consecutive blocks are vertically aligned with the same color: ● remove all 4 blocks from the board ● reset speed of falling blocks to initial speed	2
Score calculation and display	1
Check end of game when all the board is full ● Game over display ● Handling mouse event for clicking to restart the game	2

Code aesthetics (comments, readability of code, variable naming, etc.)	1

5 Submission

Submission Deadline: The deadline of this assignment is after 10 days of its release via Brightspace. No extensions will be given.

Submission Format and System: Please submit a zip file containing **all** files of your assignment, including images. Submissions via email are not accepted. Note that your solution must work using Python using Processing, otherwise your submission will not be graded.

Late Submissions: Late submissions will be penalized by 20% per 24 hours.