Odd Semester (2023)

BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:		**Surname**	**Given Names**	**Student ID Number**
	1.	**Frans**	**Christensen Mario**	**2301963316**

Course Code	**: COMP6502**	**Course Name**	**: Introduction to Programming**
Class	**: L1AC**	**Name of Lecturer(s)**	**: Bagus Manuaba**
Major	**: CS**		
Title of Assignment (if any)	: Tetris Game		
Type of Assignment	**: Final Project**		
Submission Pattern			
Due Date	**: 14-01-20**	**Submission Date**	**: 14-01-20**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Christensen Mario Frans 2301963316

COMPUTER SCIENCE PROJECT: TETRIS GAME

https://github.com/mariofrans/TetrisProject

Introduction

As a teenager, it is common that I found myself in a situation where I wished time would've accelerated, such as waiting for the internet to buffer, attaching large files on emails, or even waiting for my coffee at Starbucks. Likewise, I realized many people often meet similar situations…

Solution Design

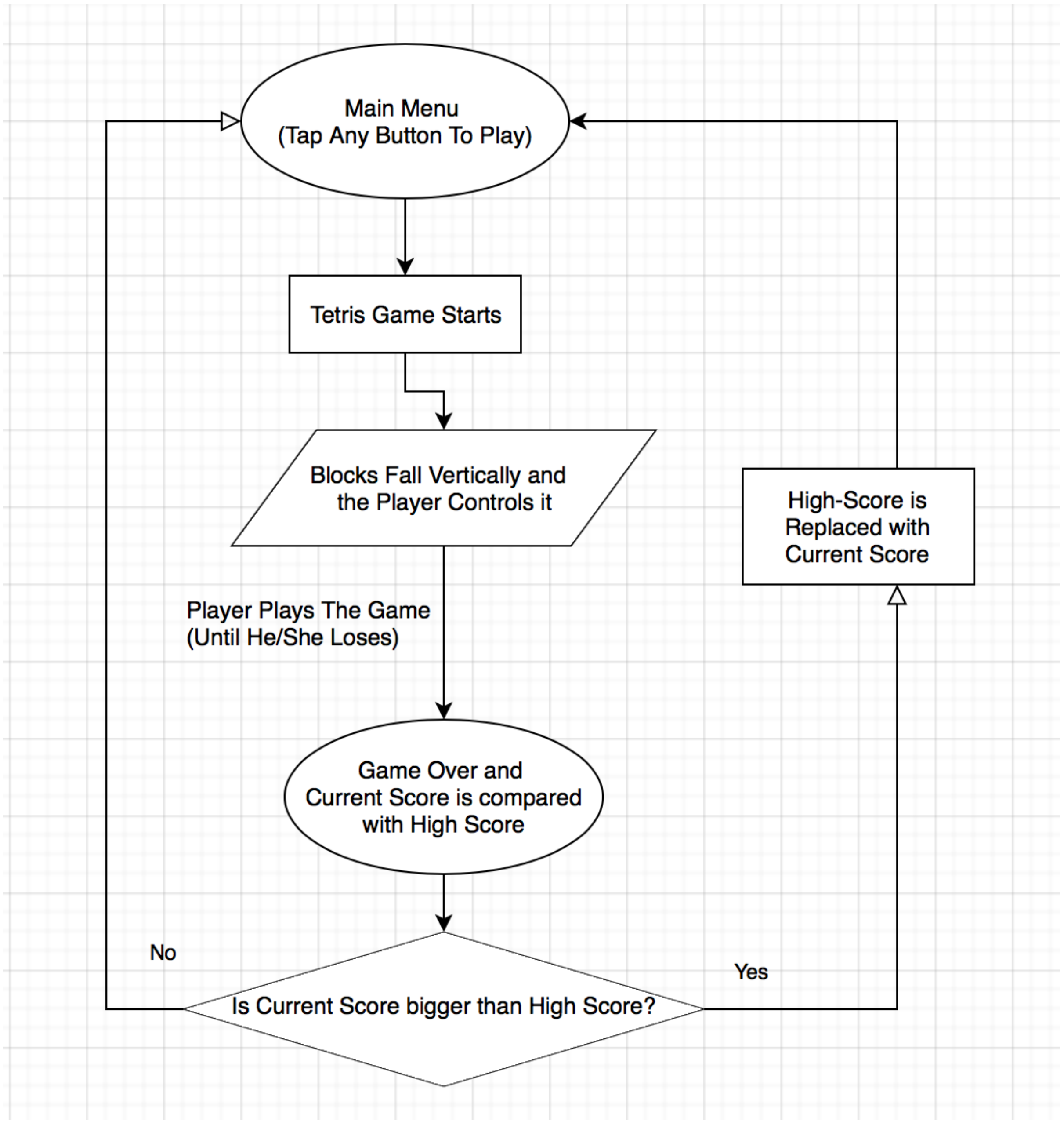
The solution I proposed for this Python project was to make a very simple time-killing computer game; Tetris. In this game, the player controls a piece of 'block', which is moving down the screen. Players have to make a row full of blocks, then, that row will disappear and the player will be rewarded with a score for each row cleared. The objective of this game is to gain as much score as possible. The player must also avoid stacking the blocks too much and not clearing them, as this makes it harder for newer spawned blocks to move around. If blocks stack up to the brim of the game screen, then the game is over and the current score will be compared with the high score; if it is bigger, then the high score will be replaced with the current score, else, nothing happens. Players will try their best to beat their own high-scores and/or compare it to other people's.

Tetris has been a very popular game worldwide for decades since its first release in 1984. It's simple yet addictive behavior catches the eyes of many, as players do not have to think much. Not only it is simple to understand for beginners, it is also quick. These are a few reasons why I suggest making a Tetris game as a solution is a perfect one. Since it is common for people to impatiently wait for time to pass by, they might as well play a enjoy a game of Tetris while waiting.

Project Specifications

For the development of this project, I'm using Python programming language for the coding, and Pycharm CE as the IDE (Integrated Development Environment. One benefit of using Python as the programming language is that the syntaxes are shorter compared to other more sophisticated programming languages, thus ending with lesser amount of lines creating similar products. Furthermore, the codes are read by the program chronologically by lines in Python, thus it is very easy to spot/avoid errors while coding and testing the program. For this game, I only use one library which is 'pygame'; the most common one for creating games in Python. This library has many several build in functions which I can use to make my codes shorter and more efficient without having to make my own.

This is a simple flowchart of the whole program:


Main Menu
(Tap Any Button To Play)
Tetris Game Starts
Blocks Fall Vertically and the Player Controls it
Player Plays The Game (Until He/She Loses)
Game Over and Current Score is compared with High Score
Is Current Score bigger than High Score?
No
Yes
High-Score is Replaced with Current Score

As seen in the flowchart above, when the player enters the program, he/she will be directed to the main menu page; which contains "Tap Any Button to Play". Once the player presses any random button, the game will start and the menu page will change into the game page. The game starts…

The controls of the game to control the blocks are "UP-button", "DOWN-button", "LEFT-button", "RIGHT-button" and "SPACE-button". For each row is filled with blocks, it will be cleared and the score will add by 10. The player plays the Tetris game until he/she loses; when the blocks reach the brim of the game screen.

Then, once the player loses the game, their score will be compared to the high score of the game, if their score is bigger,

What was implemented and how it works:

In this section, I will discuss on the algorithms implemented on this project which I learned in class.

Looping:

```
run = True
while run:
    win.fill((0, 0, 0))
    draw_text_middle(win, 'PRESS ANY KEY TO PLAY', 60, (255,255,255))
    pygame.display.update()
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
        if event.type == pygame.KEYDOWN:
            main(win)
```

In this program, I implemented several looping statements for many different purposes. For example, this while loop checks the events when the game is running, and several methods will be called. Since the player will continuously play with the game, the for-loop updates the game while the events are running. If the player quits the game(closes the window), run will be false and the game window will be closed. If the game is still running even if the player loses(game over), the program will bring them back to the main menu page so they can play again. Without this statement, the player has to close the game and relaunch it if they want to try again. Thus, this looping made the program more efficient and user friendly.

List:

```
# the shapes are put into a list
shapes = [S, Z, I, O, J, L, T]
# adjust the colors of the shapes according to the list
shape_colors = [(0, 255, 0), (255, 0, 0), (0, 255, 255), (255, 255, 0), (255, 165, 0), (0, 0, 255), (128, 0, 128)]
# index 0 - 6 represent shape
```

I also used some list in my program. These 2 lists contain the data of the different shapes. The “shape” list contains the different types of block “shapes” in Tetris (I name these block shapes according to a letter which they look like; for example: L shaped blocks looks like a letter “L”). The “shape_colors” list in the other hand contains data of the colors of the shapes above it. The indexes of these colors are according to the “shape” list, thus I know which color that I gave for each shape.

Method:

```
def draw_next_shape(shape, surface):
    font = pygame.font.SysFont('comicsans', 30)
    label = font.render('NEXT SHAPE', 1, (255,255,255))

    sx = top_left_x + play_width + 50
    sy = top_left_y + play_height/2 - 100
    format = shape.shape[shape.rotation % len(shape.shape)]

    pygame.draw.rect(surface, shape.color, (sx + j*block_size, sy + i*block_size,    block_size, block_size), 0)

    surface.blit(label, (sx + 10, sy - 30))
```

There are also several methods which I used for running the game. For example, the method above is responsible for informing the player the “next shape” which will be spawned. As seen the method has 2 parameters; the shape and its surface(color) which is chosen at random when this method is called. To make it user friendly, I added a label on top of it which shows “NEXT SHAPE”, to make it clear what it is showing to the player. Then, I adjust the position of this “image” (Since the game show an image of the next shape) on the screen. Then the codes below will do the drawing of this image.

Class:

```
class Piece(object):  # *
    def __init__(self, x, y, shape):                    # initializes the class
        self.x = x
        self.y = y
        self.shape = shape                              # index of the list
        self.color = shape_colors[shapes.index(shape)]      # calls the list using the index 'shape'
        self.rotation = 0                           # default rotation is 0
```

Class UML:

Piece
- x - y - shape - color - rotation

For this game, there is one class ‘Piece’, which is responsible for creating the pieces of the blocks(objects) to be spawned. The x and y values are the coordinates of the object, the shape is chosen by random from the list as well as its color.

Libraries:

```
import pygame
```

There is only one library I used in this project which is Pygame

Bool:

```
change_piece = False
run = True
```

These are some Booleans which I have in the code. The “change_piece” is False, meaning that the piece of block which the player is controlling will not change until the next shape is spawned. If the next shape is spawned, then the new piece which the player will be controlling will change to the newly spawned one. “run” will always be True as long as the game is running, and will be False if the player quits the game.

If & Else:

```
# configuring the buttons and its function in the game
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_LEFT:
        current_piece.x -= 1
        if not(valid_space(current_piece, grid)):
            current_piece.x += 1
    if event.key == pygame.K_RIGHT:
        current_piece.x += 1
        if not(valid_space(current_piece, grid)):
            current_piece.x -= 1
    if event.key == pygame.K_DOWN:
        current_piece.rotation += 1
        if not (valid_space(current_piece, grid)):
            current_piece.rotation -= 1
    if event.key == pygame.K_SPACE:
        fall_speed = fall_speed - 10000
    if event.key == pygame.K_UP:
        current_piece.rotation += 1
        if not(valid_space(current_piece, grid)):
            current_piece.rotation -= 1
```

These are some of the if-statements; including nested if-statements which I included in my program. In the above code, the if-statements checks for the buttons that the player presses on his/her computer, and executes what will happen to the controlled object in the game. For example, if the player presses the "left button" on their computer while the game is running, the moving Tetris piece will go to the left.

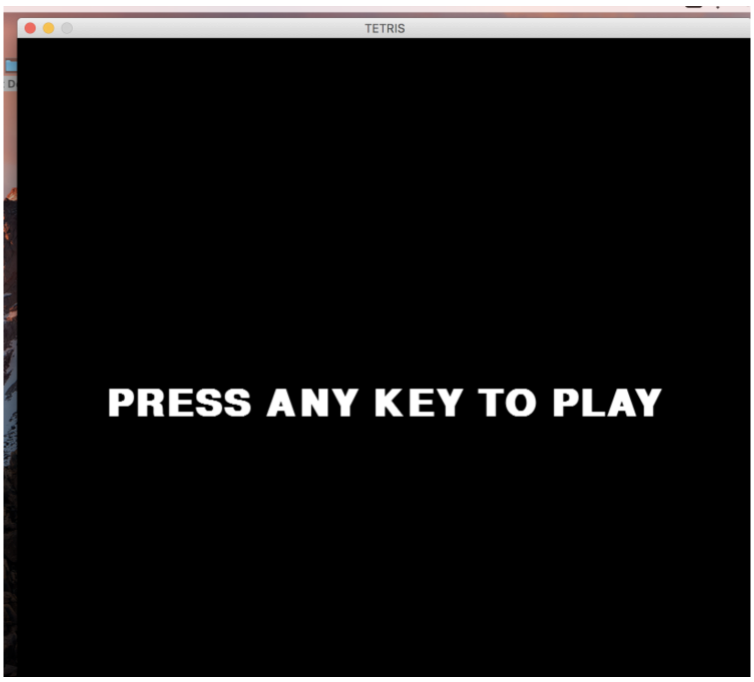
Other Implementations:

Aside from the above implementations which I have for my program, there are also other minor implementations which I have added on to this program which I learned in class. One of which are the mass use of comments. I used a lot of comments in the program as I find it very useful to remind me what the code that I created does. Without them, I might even forget what some the codes are supposed to be. It also makes other programmers understand my code more easily in case I need help with my codes.

Evidence of the Working Program(Including Screenshots):

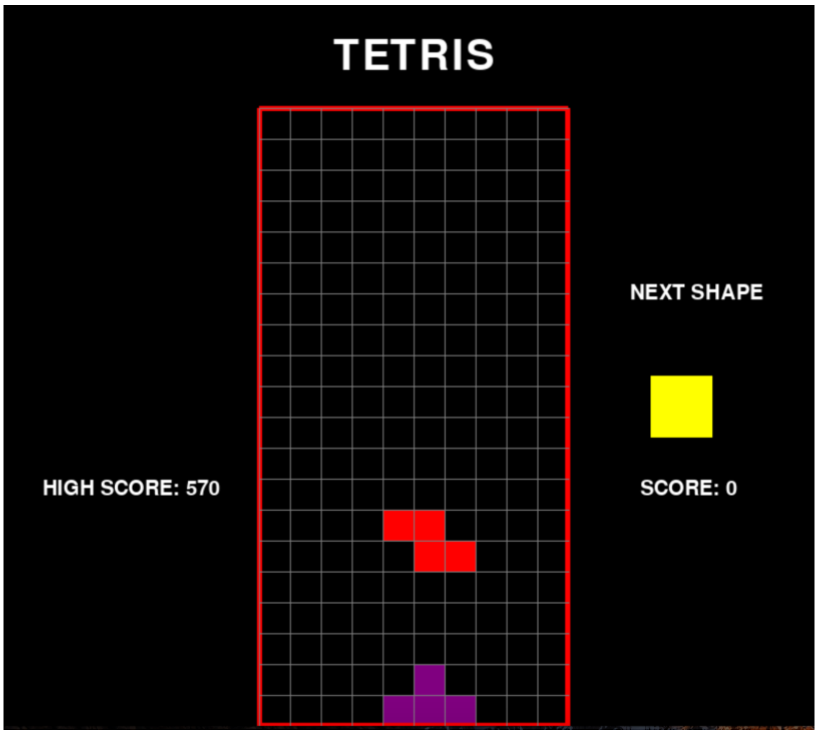
(Disclaimer: since Tetris is a high-speed game, it is not possible to take screenshots of some specific events as things happen too fast. Thus, the screenshots gathered below are just basic ones. To fully test and proof the interface of the program, it is recommended run the python code.)

1. This screenshot shows the main menu of the game, to keep it simple and user friendly, I only have "Tap anywhere to start game" in the middle of the screen:

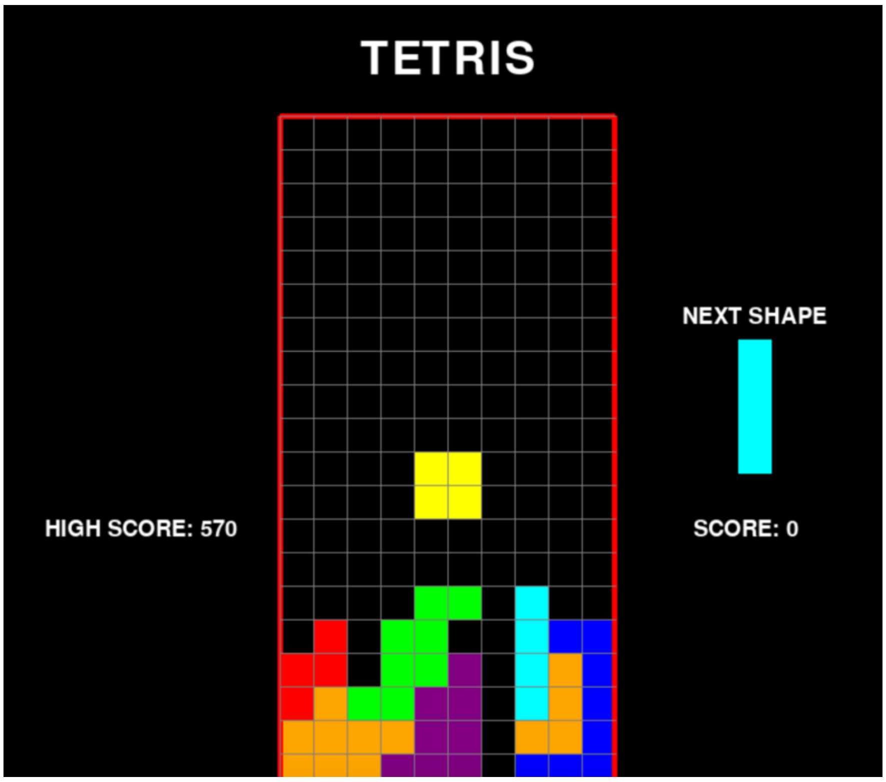


2. The next screenshot shows the blocks moving down the screen while the player is controlling it; blocks are spawned on the red border right below the 'TETRIS' title:

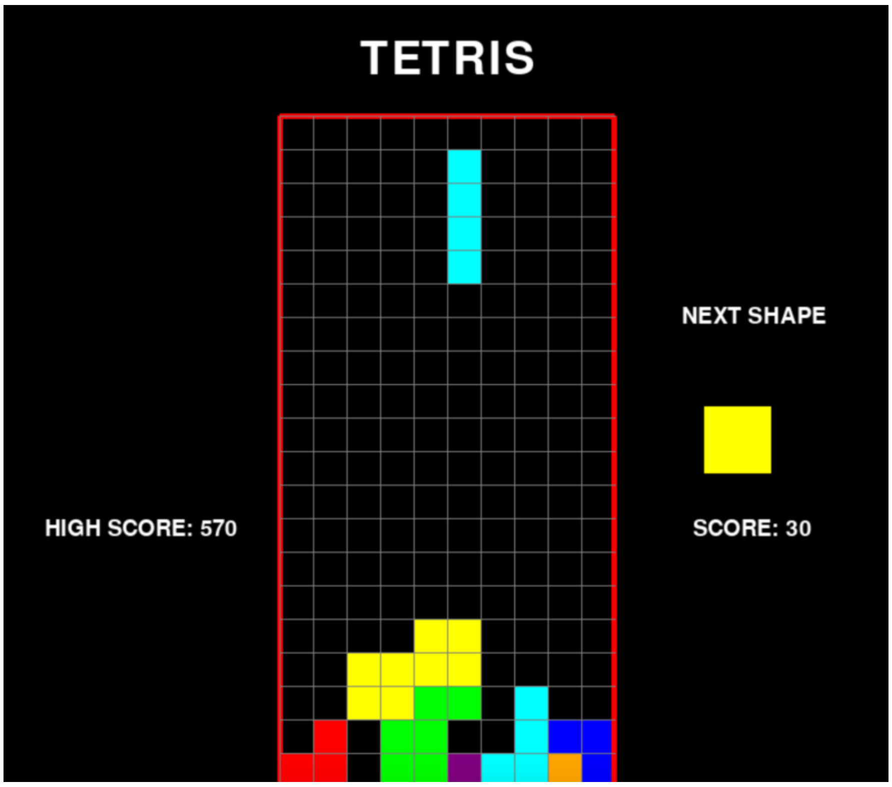
As seen in the display, there is the high-score on the left (to tell players what are their high scores to beat), their current score on the right, and the 'NEXT SHAPE' which tells the player what will be the next shape to be spawned when the current block he/she is controlling already touches the bottom/another block and can no longer be moved.



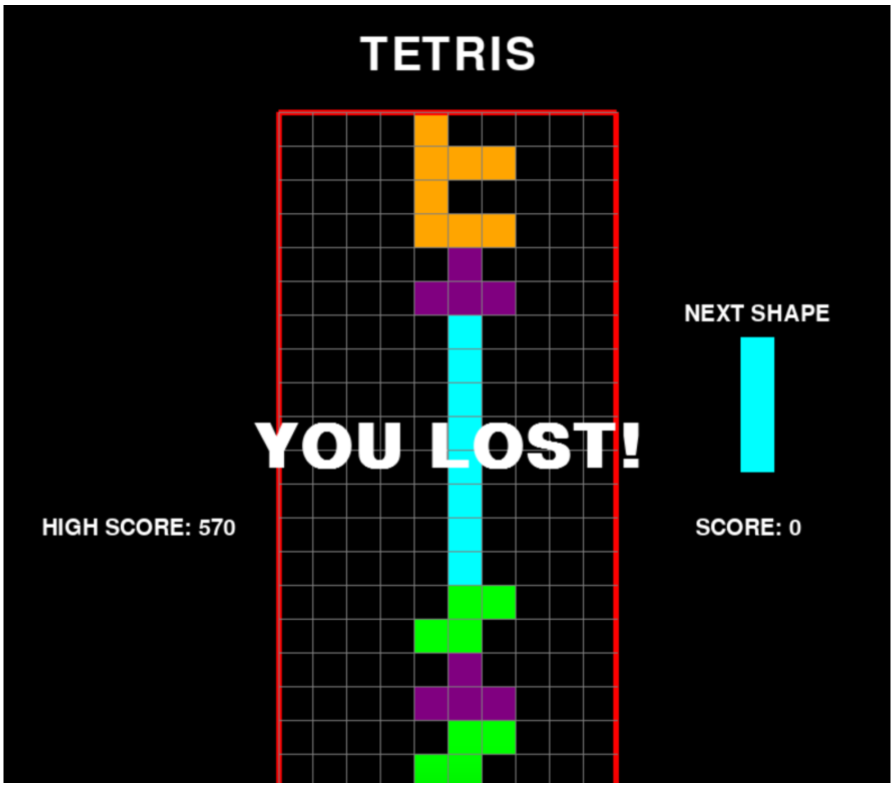
3. The next screenshot shows the piles of block which I have piled, as seen in the score, I haven't had any rows cleared, thus it is zero. (I purposely stack the blocks in this manner to prove in the next screenshot that when the rows are cleared, we can see the differences; what happens to the 'completed' rows and what happens to the 'incomplete' rows):



4. In this screenshot, I have cleared 3 rows from the previous screenshot, by putting the long i-shaped block vertically in the 7th column of the game area. As proven in this and the previous screenshots, the piles that are completed had already disappear, and the piles on top of them; which are incomplete is dragged down the screen. The game continues…



5. Finally, if this screenshot shows what happens when the player consistently fails to clear the rows, making the blocks pile up to the brim; where new blocks are spawned. In this case, newer blocks cannot be spawned, and the game is over. The current score will then be compared to the high score; if the current score is bigger, then the high score will be replaced with the current score. This screenshot proves that the code runs properly when the game is over, showing "YOU LOST!", then after 1.5 seconds, it will go back to the main menu page.
(*Note: I purposely run the game only to stack up the blocks to show the "YOU LOST!" screenshots*)



(*Note: These screenshots are just a few events of the game, for more and complete details of the running game, please refer to the screen-recorded video of myself playing the game which I will be including in the submission.*)

References:
https://techwithtim.net/tutorials/game-development-with-python/tetris-pygame/tutorial-3/

For this project, I used the following codes which I found on the internet.

1. This is the "SHAPE FORMAT" section, which I used to initiate the shapes of the Tetris blocks. As seen in the codes below, the letters variable represent the shape of the Tetris blocks which it look like. For example, the "S" variable represents the blocks which look like an "S" letter. Below that are the different rotations which each block can have, for example the "S" shaped blocks can only have 2 possible rotations while the "J" shaped blocks have 4 different possible rotations. Below these codes, I put all the shape formats in a list called "shapes", and also its colors in "shape_colors" with their respective indexes.

```
# SHAPE FORMATS

S = [['.....',
      '......',
      '..00..',
      '.00...',
      '.....'],
     ['.....',
      '..0..',
      '..00.',
      '...0.',
      '.....']]

Z = [['.....',
      '.....',
      '.00..',
      '..00.',
      '.....'],
     ['.....',
      '..0..',
      '.00..',
      '.0...',
      '.....']]

I = [['..0..',
      '..0..',
      '..0..',
      '..0..',
      '.....'],
     ['.....',
      '0000.',
      '.....',
      '.....',
      '.....']]

O = [['.....',
      '.....',
      '.00..',
      '.00..',
      '.....']]

J = [['.....',
      '.0...',
      '.000.',
      '.....',
      '.....'],
     ['.....',
      '..00.',
      '..0..',
      '..0..',
      '.....'],
     ['.....',
      '.....',
      '.000.',
      '...0.',
      '.....'],
     ['.....',
      '..0..',
      '..0..',
      '.00..',
      '.....']]

L = [['.....',
      '...0.',
      '.000.',
      '.....',
      '.....'],
     ['.....',
      '..0..',
      '..0..',
      '..00.',
      '.....'],
     ['.....',
      '.....',
      '.000.',
      '.0...',
      '.....'],
     ['.....',
      '.00..',
      '..0..',
      '..0..',
      '.....']]

T = [['.....',
    '..0..',
    '.000.',
    '.....',
    '.....'],
   ['.....',
    '..0..',
    '..00.',
    '..0..',
    '.....'],
   ['.....',
    '.....',
    '.000.',
    '..0..',
    '.....'],
   ['.....',
    '..0..',
    '.00..',
    '..0..',
    '.....']]

shapes = [S, Z, I, O, J, L, T]
shape_colors = [(0, 255, 0), (255, 0, 0), (0, 255, 255), (255, 255, 0), (255, 165, 0), (0, 0, 255), (128, 0, 128)]


# index 0 - 6 represent shape
```

2. This function works to create the grids of the game area where the tetris blocks are moving around. There are 10 columns and 20 rows, therefore the grid is set up with a "for-loop"; "in range(10)" and in "range(20)" respectively. These are the black tiles which are not occupied by the blocks, therefore its color is written as "(0, 0, 0)" which represents black color.

```
def create_grid(locked_pos={}):  # *

  # creates list in a list for the BLANK grids
  # so 20 rows(sublist) for each column(list)
  # each column has 10 grids

  grid = [[(0,0,0) for _ in range(10)] for _ in range(20)]      # the grid is initially black
```

```
# register grids that are already filled with blocks
# to check every position if it is filled with blocks

for i in range(len(grid)):
    for j in range(len(grid[i])):
        if (j, i) in locked_pos:            # 'j' are x-values(column), 'i' are y-values(rows)
            c = locked_pos[(j,i)]            # locked_pos is a dictionary for the position & color, keys & values
            grid[i][j] = c                   # to set the color of the grid
return grid
```

3. This piece of code updates the "fall_time", which is the time for the blocks to move down each grid. It uses "clock.tick()" which is a built in function in Python to execute the counting of the time.

```
# gets the real time
fall_time += clock.get_rawtime()
clock.tick()
```