Algorithm and Programming
Final Project

Project Name: Tetris

Name: Vania Agnes Djunaedy
Student ID: 2602158531
Class: L1BC

Table of Contents

A. Description

I. Introduction to the program

So we were told our schedule ahead of time and that in week 13, we would be doing our final project after thinking about what would be best. I suddenly thought of doing Tetris. Now, if you don't know what Tetris is, Tetris is one of the most popular and recognizable video games of all time and has been released on numerous platforms.

However, I want to modify it to be different from the Tetris you find on the internet. I changed its color, making it cuter and less tiresome than regular Tetris.

II. The function of the program

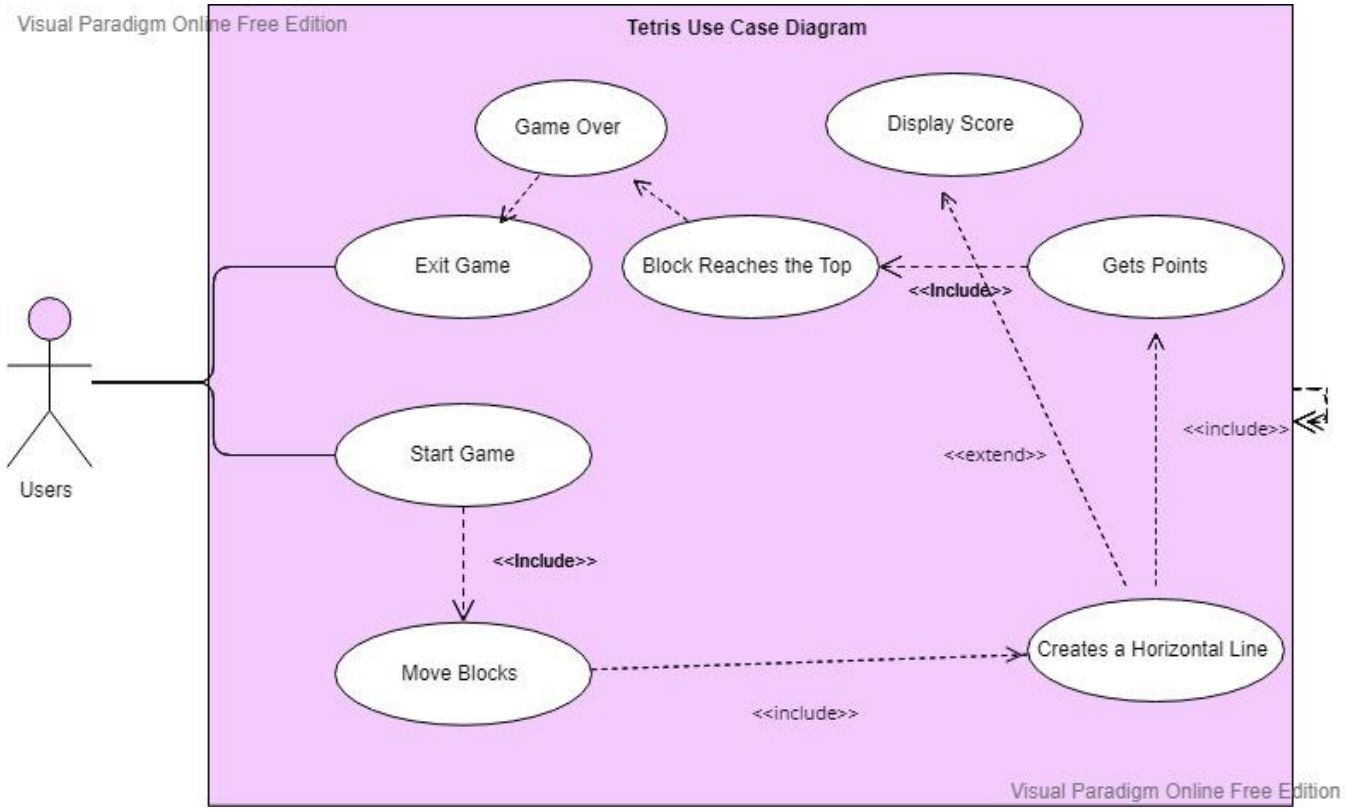
The main reason why I made this program was to entertain more people. People who find the standard colors in Tetris boring may be more interested in the Tetris I make.

Tetris is a puzzle game in which the player must manoeuvre falling tetrominoes, which are geometric shapes made of four square blocks each, to create horizontal lines without gaps. The pieces will fall from the top, and players can move to the left and right and rotate the shapes as they fail to fit them in the existing game board pattern.

The game's objective is to clear the lines by completing the horizontal rows without any gaps. Every time a line is completed, it disappears, and the player earns points. The game will continue until the stacks of pieces reach the top. The goal of this game is to get as many points before the piles reach the top.

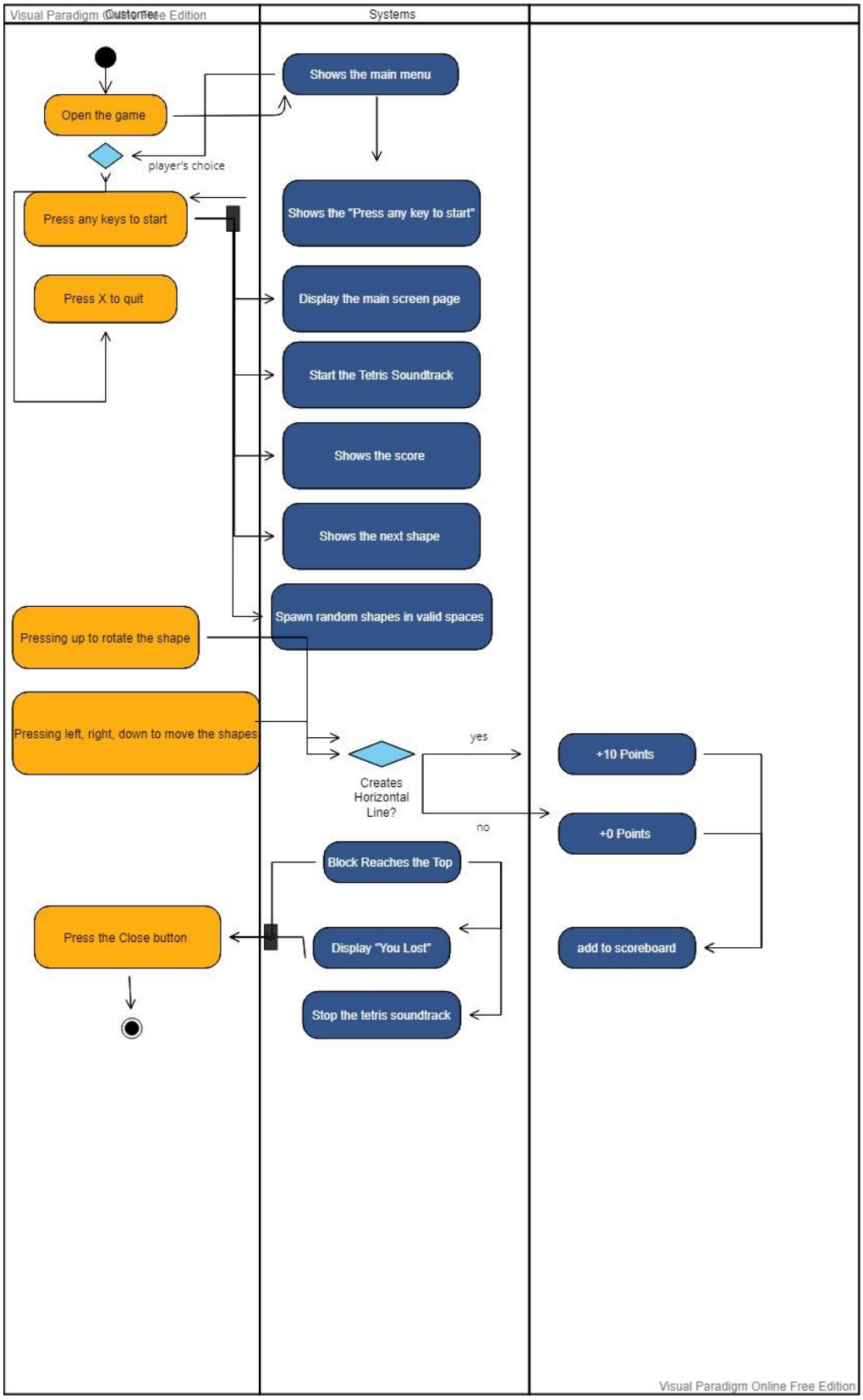
The game increases their difficulty as time increases. The more time you spent on the game, the faster the pieces would fall.

B. Use-Case Diagram



The diagram above shows the Use Case Diagram for Tetris.

C. Activity Diagram



The diagram above shows the Activity Diagram for Tetris.

D. Class Diagram

Pieces
-x -y -shape -color -rotation
+ creating_grid(locked pos={}) +conversion_shape_format(shape) +valid_space(shape, grid) +check_lost(positions) +get_the_shape() +text_middle(surface, text, size, color) +drawing_grid(surface, grid) +clearing_rows(grid, locked) +next_shape(shape,surface) +update_scores(nscore) +maximum_scores() +drawing_window(surface, grid, score=0, last_score = 0) +main(win) +main_menu(win)

This above is the class diagram of Tetris.

E. Modules

I. Importing Pygame

In order to write video games, Pygame was one of the first modules I utilized when I started coding. In addition to many other capabilities used in games, it offers various features for things like playing music, handling events, and handling visuals. To use it simply write "*import pygame"*

II. Importing Mixer

The mixer module from PyGame was the second module I utilized. In the pygame program, the mixer module is used to play and control sounds and music. It supports a number of sound formats, including WAV, OGG, and even MP3. However, I utilized a WAV Format one for my coding. To use it simply write "*import pygame"* in your coding followed by "*from pygame import mixer".*

III. Importing Random Module

The random module in python provides a function for generating random numbers or operations in python. To use it simply write "*import random"*. It served as a tool for Tetris' shape randomization.

F. Essential Algorithms

1. Making the Class

```
class Piece(object):
    def __init__(self, x, y, shape):
        self.x = x
        self.y = y
        self.shape = shape
        self.color = shape_colors[shapes.index(shape)] #
        self.rotation = 0
```

This basically bundles the data needed later on for the codings. The class I made was heavily used and mentioned throughout the coding.

2. Creating the grid

```
def creating_grid(locked_pos={}):  #finding the corresponding
    grid = [[(0,0,0) for _ in range(10)] for _ in range(20)] #
```

This function is to make the grid. This function is used to make a list for each of the row, and find the corresponding position to the locked position to get an accurate grid.

3. Converse the shape format

```
def conversion_shape_format(shape):
    positions = []
    format = shape.shape[shape.rotation % len(shape.shape)]

    for v, line in enumerate(format):
        row = list(line)
        for w, column in enumerate(row):
            if column == '0':
                positions.append((shape.x + w, shape.y + v))

    for v, pos in enumerate(positions):
        positions[v] = (pos[0] - 2, pos[1] - 4)

    return positions
```

This function is created so that the computer could understand the list that we created.

4. Valid Spaces

```
def valid_spaces(shape, grid):
    accepted_pos = [[(w, v) for w in range(10) if grid[v][w] == (0,0,0)] for v in range(20)]
    accepted_pos = [w for sub in accepted_pos for w in sub]
    #colours other than black means they are occupied, if not then free.

    formatted = conversion_shape_format(shape)

    for pos in formatted:
        if pos not in accepted_pos:
            if pos[1] > -1:
                return False
    return True
```

This function is created so that the shapes would be moving in the valid spaces, and to ensure so we made the coding so that if the colors is other than black that means it's occupied and if's it black then it's free

5. Check Lost

```
def check_lost(positions):
    for pos in positions:
        x, y = pos
        if y < 1:
            return True

    return False  #if the users
```

This function is created to see if the players lost the game, if they lost then the game will stop.

6. Get Shape

```
#to generate a random shape during the game
def get_the_shape():
    return Piece(5, 0, random.choice(shapes))
```

This function is to make sure that it generates random shapes during the game

7. Middle Text used in Games

```
#display text to the middle of the screen during main menu and end screen of the game
def text_middle(surface, text, size, color):
    font = pygame.font.SysFont("comicsans", size, bold=True)
    label = font.render(text, 1, color)

    surface.blit(label, (tlp_x + block_width /2 - (label.get_width()/2), tlp_y + block_height/2 - label.get_height()/2))
```

This function is used to display the text in the middle screen when the game starts and when the users lost the game.

8. Drawing the grid

```
#this function is the one that draws the grey lines that we seen in the game
def drawing_grid(surface, grid):
    sx = tlp_x
    sy = tlp_y

    for v in range(len(grid)): #horizontal lines
        pygame.draw.line(surface, (128,128,128), (sx, sy + v*block_size), (sx+block_width, sy+ v*block_size))
        for w in range(len(grid[v])): #vertical lines
            pygame.draw.line(surface, (128, 128, 128), (sx + w*block_size, sy),(sx + w*block_size, sy + block_height))
```

This function is created so that we can draw the gray lines used as the grid in the games.

9. Clearing the rows

```
#function to check if the row is full, if it's full then it would be deleted
def clearing_rows(grid, locked):
# needs to see whether the row is clear the shift every other row above down one
    inc = 0
    for v in range(len(grid)-1, -1, -1):
        row = grid[v]
        if (0,0,0) not in row:
            inc += 1
            #adding position to remove from the locked
            ind = v
            for w in range(len(row)):
                try:
                    del locked[(w,v)]
                except:
                    continue

    if inc > 0:
        for key in sorted(list(locked), key=lambda x: x[1])[::-1]:
            x, y = key
            if y < ind:
                newKey = (x, y + inc)
                locked[newKey] = locked.pop(key)

    return inc
```

This function is used so that if a player has fulfilled the requirement of making a horizontal line then the row would be deleted then they would add one.

10. Displaying next shape

```
def next_shape(shape, surface):
    font = pygame.font.SysFont('comicsans', 30)
    label = font.render('Next Shape', 1, (0,0,0))

    sx = tlp_x + block_width + 50
    sy = tlp_y + block_height/2 - 100
    format = shape.shape[shape.rotation % len(shape.shape)]

    for v, line in enumerate(format):
        row = list(line)
        for w, column in enumerate(row):
            if column == '0':
                pygame.draw.rect(surface, shape.color, (sx + w*block_size, sy + v*block_size, block_size, block_size), 0)

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

This is used so that we could see the next shapes that is going to fall down

11. Update Scores

```
#to update the scores
def update_scores(nscore):
    score = maximum_scores()

    with open('scores.txt', 'w') as f:
        if int(score) > nscore:
            f.write(str(score))
        else:
            f.write(str(nscore))
```

This is used to update the scores shown in the right side of the screen

12. Drawing Window

```
#displaying the current and best score
def drawing_window(surface, grid, score=0, last_score = 0):
    surface.fill((255, 255, 255))
    #Tetris Title
    pygame.font.init()
    font = pygame.font.SysFont('comicsans', 60)
    label = font.render('Tetris', 1, (0, 0, 0))

    surface.blit(label, (tlp_x + block_width / 2 - (label.get_width() / 2), 30))

    # current score
    font = pygame.font.SysFont('comicsans', 30)
    label = font.render('Score: ' + str(score), 1, (0, 0, 0))

    sx = tlp_x + block_width + 50
    sy = tlp_y + block_height/2 - 100

    surface.blit(label, (sx + 20, sy + 160))
    # last score
    label = font.render('High Score: ' + last_score, 1, (255, 255, 255))

    sx = tlp_x - 200
    sy = tlp_y + 200

    surface.blit(label, (sx + 20, sy + 160))

    for v in range(len(grid)):
        for w in range(len(grid[v])):
            pygame.draw.rect(surface, grid[v][w], (tlp_x + w*block_size, tlp_y + v*block_size, block_size, block_size), 0)

    pygame.draw.rect(surface, (255, 0, 0), (tlp_x, tlp_y, block_width, block_height), 5)

    drawing_grid(surface, grid)
    #pygame.display.update()
```

This is used to display the current and best score. And also for the format of the letters.

13. Main Menu

```
                update_scores(score)
                mixer.music.stop() #the music stops

#when the game launched, the users can press any button to start the game
def main_menu(win):
    run = True
    while run:
        win.fill((0,0,0))                                    255
        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)

    pygame.display.quit()
```

When the game is launched it would show the text “ press any buttons to start”

14. Main Algorithm

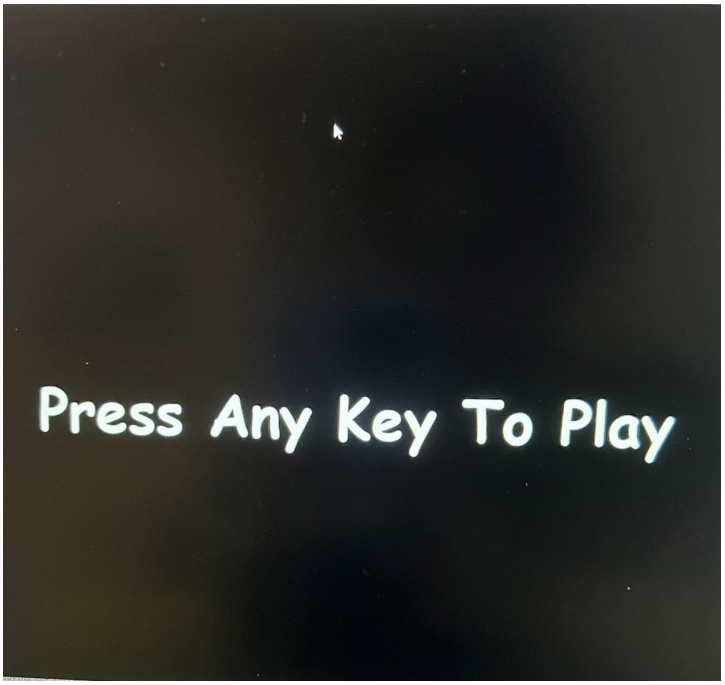
```
def main(win):
    last_score = maximum_scores()
    locked_positions = {}
    grid = creating_grid(locked_positions)

    change_piece = False
    run = True
    current_piece = get_the_shape() #gets the current shape
    next_piece = get_the_shape() #gets next shape
    clock = pygame.time.Clock()
    fall_time = 0
    fall_speed = 0.27
    level_time = 0
    score = 0 #the game starts with 0

    #as time goes on, the games get more difficult
    while run:
        grid = creating_grid(locked_positions)
        fall_time += clock.get_rawtime()
        level_time += clock.get_rawtime()
        clock.tick()

        if level_time/1000 > 5:
            level_time = 0
            if level_time > 0.12:
                level_time -= 0.005

        #piece falling codes
        if fall_time/1000 > fall_speed:
            fall_time = 0
            current_piece.y += 1
            if not(valid_spaces(current_piece, grid)) and current_piece.y > 0:
                current_piece.y -= 1
                change_piece = True
```

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        run = False
        pygame.display.quit()

    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_LEFT: #move shape to the left
            current_piece.x -= 1
            if not(valid_spaces(current_piece, grid)):
                current_piece.x += 1
        if event.key == pygame.K_RIGHT: #move shape to the right
            current_piece.x += 1
            if not(valid_spaces(current_piece, grid)):
                current_piece.x -= 1
        if event.key == pygame.K_DOWN: #move shape down
            current_piece.y += 1
            if not(valid_spaces(current_piece, grid)):
                current_piece.y -= 1
        if event.key == pygame.K_UP: #rotates the shape
            current_piece.rotation += 1
            if not(valid_spaces(current_piece, grid)):
                current_piece.rotation -= 1

shape_pos = conversion_shape_format(current_piece)
#adding colour of the piece to the grid for drawing
for v in range(len(shape_pos)):
    x, y = shape_pos[v]
    if y > -1:
        grid[y][x] = current_piece.color
#if the piece hits the grounds
if change_piece:
    for pos in shape_pos:
        p = (pos[0], pos[1])
        locked_positions[p] = current_piece.color
    current_piece = next_piece
    next_piece = get_the_shape()
    change_piece = False
    score += clearing_rows(grid, locked_positions) * 10

drawing_window(win, grid, score, last_score)
next_shape(next_piece, win)
pygame.display.update()
```

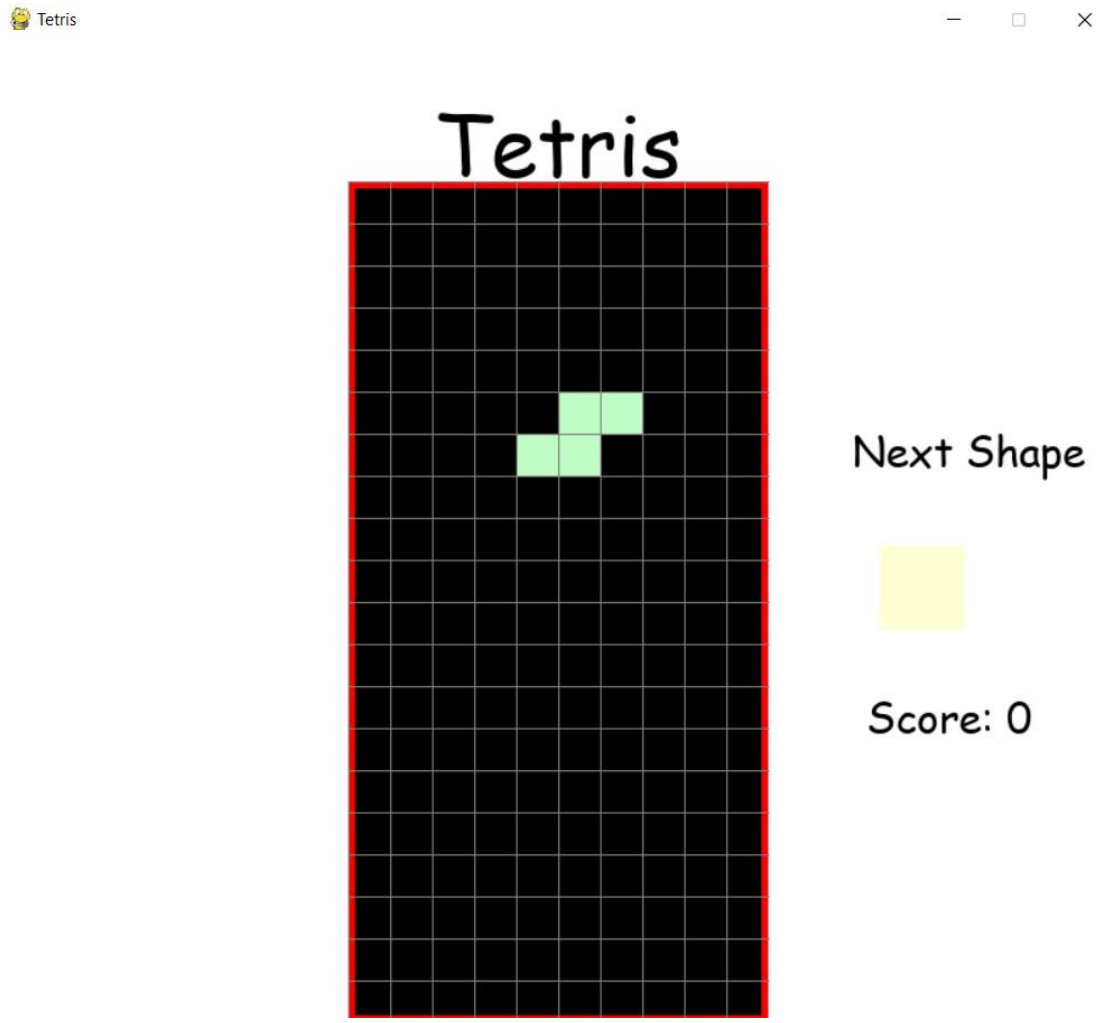
```
#checks if the users has lost
if check_lost(locked_positions):
    text_middle(win, "YOU LOST!", 80, ())
    pygame.display.update()
    pygame.time.delay(1500)
    run = False
    update_scores(score)
    mixer.music.stop() #the music stops
```

This is basically where all the main algorithm runs in and this is a very important aspect in the coding.

G. Screenshot of your application

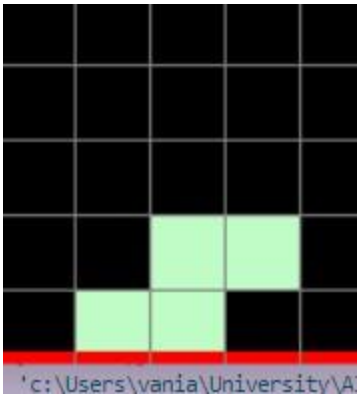


This is the before the game starts or the main menu

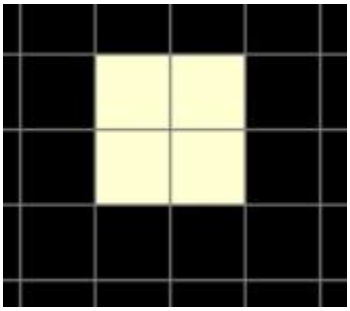


This is the visual when we played the game as we can see there is many writing like the score next shape and etc.

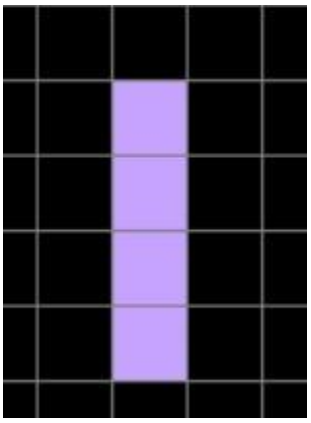
This is the visualization when we lost the game



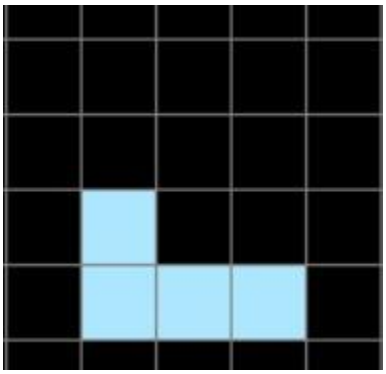
This is the S shape of the Tetris



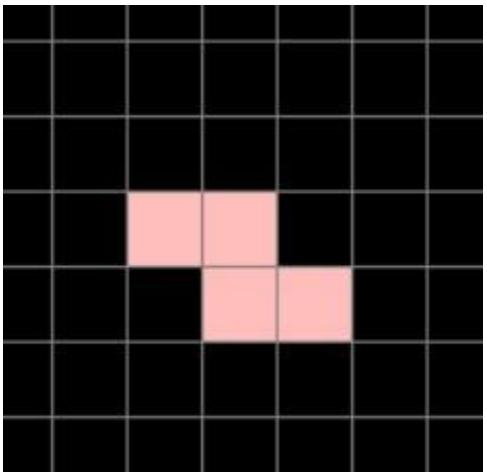
This is the O shape of the tetris



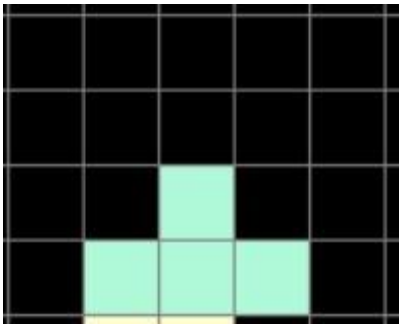
This is the I shape of the tetris



This is the L shape of the tetris



This is the Z shape



This is the T shape

H. Lesson Learned/ Reflection

I. Pygame

From here I learned how to install pygame and implement it with python. I also learned how to use the other modules provided by pygame. I also learned that pygame can be used to implement game

II. Importing Mixer

I wanted to input music on my Tetris so I decided to research and with that I gained more knowledge and now I know how to add music in python but using the mixer module I can now add music to the game.

III. Coding for a game

This is my first time making codings for a game and this has been a life experience to say the least, definitely the hardest project i worked so far in python, and definitely the most impactful.

Reflection

Making this project taught me many new skills, including how to incorporate random, PyGame, and Mixer into my coding. I also learned how to create Tetris using all the knowledge I gained in class. I mastered the art of music input for coding. I also picked up some new skills, like how to input grids and how to remove lines when blocks generate horizontal lines. The right side of the screen has instructions on how to make the system save results and data as well as a system that displays the next shapes. So, to put it mildly, my process for creating this has been educational and inspirational.