PROJECT REPORT: TETRIS

SPECIFICATIONS

Objective

Create a simple score based system ‘Tetris’ game using pygame, with base controls, and with the addition of simple UI sufficient to bring comfort to players’ gameplay.

Description

Much like any other ‘Tetris’ game, this game allows players to control the falling blocks, with the goal of preventing the stacked blocks to reach the top. In order to both attain points and remove the layer of blocks, they must align 10 blocks horizontally on the grid. The players will have to fight against endless waves of ‘Tetris’ blocks and attain a highscore. The controls uses simple arrow keys to adjust the falling speed, rotation, & moving the blocks right/left.

The Design

Files: Tetris.py, scores.txt, 1.png, 2.png, 3.png, 4.png, background.jpeg, ES_Caracal - Leah Ryder.mp3, ES_Magic Chime - SFX Producer.wav, ES_Video Game Descend 8 - SFX Producer.wav

Tetris.py

The Tetris.py contains the main function, initializing variable datas, menu, & game functions.

```python
import pygame
import random

pygame.font.init()
pygame.init()


s_width = 800
s_height = 700
play_width = 300
play_height = 600
block_size = 30
img1_old = pygame.image.load(r'1.png')
img1 = pygame.transform.scale(img1_old, (30, 30))
img2_old = pygame.image.load(r'2.png')
img2 = pygame.transform.scale(img2_old, (30, 30))
img3_old = pygame.image.load(r'3.png')
img3 = pygame.transform.scale(img3_old, (30, 30))
img4_old = pygame.image.load(r'4.png')
img4 = pygame.transform.scale(img4_old, (30, 30))
bg_img = pygame.image.load(r'background.jpeg')
row_clear_sound = pygame.mixer.Sound("ES_Magic Chime - SFX Producer.wav")
row_clear_sound.set_volume(0.5)
lose_sound = pygame.mixer.Sound("ES_Video Game Descend 8 - SFX Producer.wav")
pygame.mixer.music.load("ES_Caracal - Leah Ryder.mp3")
pygame.mixer.music.set_volume(0.1)

top_left_x = (s_width - play_width) // 2
top_left_y = s_height - play_height


```

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

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

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

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

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

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

```
114     T = [['.....',
115          '..0..',
116          '.000.',
117          '.....',
118          '.....'],
119         ['.....',
120          '..0..',
121          '..00.',
122          '..0..',
123          '.....'],
124         ['.....',
125          '.....',
126          '.000.',
127          '..0..',
128          '.....'],
129         ['.....',
130          '..0..',
131          '.00..',
132          '..0..',
133          '.....']]
134
135     shapes = [S, Z, I, O, J, L, T]
136     shape_colors = [img1, img2, img3, img4, img4, img2, img3]
137
```

Initializations

All the images above shows the initialization of important variables that are later used in the functions to run the game.

- *windows dimension:* line 8 - 12 + line 28 - 29; this includes the dimension for the Tetris playing grid, each square dimension in the grid, the width and length at the side of the grid.

- *block & background image:* line 13 - 21; this stores the imported image for the background and the coloured block 30 x 30 for the grid.

- *sound effects:* line 22-26; import the sound effects for the game and the background song.

- *shapes & colours:* line 33-136; stores the shape for the 'Tetris' blocks & the colour it for it from the imported image.

```
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

```

Piece class

The Piece is a class because of its necessity & convenience being used as one by the other functions later on. It will represent the shape of the block being taken, the colour, and its rotation, as well as coordinates.

```

def create_grid(locked_pos={}):
    grid = [[(0,0,0) for _ in range(10)] for _ in range(20)]

    # len(grid) is the row on the grid (y-axis), len(grid[i]) is the column (x-axis)
    for i in range(len(grid)):
        for j in range(len(grid[i])):
            if (j, i) in locked_pos:
                c = locked_pos[(j,i)]
                grid[i][j] = c
    return grid
```

create_grid function

This function is the base for the game's grid, it is where the blocks are displayed and the game is played. It is 20 block_size vertically and 10 block_size horizontally (1 block_size = 30px).

```
209
210 def draw_grid(surface, grid):
211     sx = top_left_x
212     sy = top_left_y
213
214     # len(grid) is the row on the grid (y-axis), len(grid[i]) is the column (x-axis)
215     for i in range(len(grid)):
216         pygame.draw.line(surface, (128,128,128), (sx, sy + i*block_size), (sx+play_width, sy+ i*block_size))
217         for j in range(len(grid[i])):
218             pygame.draw.line(surface, (128, 128, 128), (sx + j*block_size, sy),(sx + j*block_size, sy + play_height))
219
```

draw_grid function

After creating the grid, this function will create the grid tiles by making the border of the blocks white.

```
198
199 def get_shape():
200     return Piece(5, 0, random.choice(shapes))
201
202
203 def draw_text_middle(surface, text, size, color):
204     font = pygame.font.SysFont("couriernew", size, bold=True)
205     label = font.render(text, 1, color)
206
207     surface.blit(label, (top_left_x + play_width /2 - (label.get_width()/2), top_left_y + play_height/2 - label.get_height()/2))
208
```

get_shape function

A simple function that will randomly pick between the 7 shapes available (This will also determine its colour).

draw_text_middle function

This is a function made simply for ease of use, it will create a centred text with desired size, text, and colour.

```
220
221 def clear_rows(grid, locked):
222
223     inc = 0
224     for i in range(len(grid)-1, -1, -1):
225         row = grid[i]
226         if (0,0,0) not in row:
227             inc += 1
228             ind = i
229             pygame.mixer.Sound.play(row_clear_sound)
230             for j in range(len(row)):
231                 try:
232                     del locked[(j,i)]
233                 except:
234                     continue
235
236     if inc > 0:
237         for key in sorted(list(locked), key=lambda x: x[1])[::-1]:
238             x, y = key
239             if y < ind:
240                 newKey = (x, y + inc)
241                 locked[newKey] = locked.pop(key)
242
243     return inc
```

clear_rows function

This function is used to both check and remove a row once its row of 10 is filled with full coloured blocks. Aside from that it will also open the locked rows of block above the row being removed, so that it will be able to descend one block down. Since the row was deleted, the row above it will replace the row that was deleted this way.

```
245
246 def draw_next_shape(shape, surface):
247     z = 0
248     num = 0
249     ex = [30, 15, 0, -15, -20]
250     font = pygame.font.SysFont('couriernew', 30, bold = True)
251     label = font.render('Next Shape', 1, (255,255,255))
252
253     sx = top_left_x + play_width + 50
254     sy = top_left_y + play_height/2 - 100
255     format = shape.shape[shape.rotation % len(shape.shape)]
256
257     for i, line in enumerate(format):
258         if line == 0[0][i]:
259             z=z+1
260             num = z-1
261
262     for i, line in enumerate(format):
263         row = list(line)
264         for j, column in enumerate(row):
265             if column == '0':
266                 win.blit(shape.color, (sx + j * block_size + 15,sy + i * block_size + ex[num]))
267
268     surface.blit(label, (sx + 10, sy - 30))
269
```

draw_next_shape function

This function will be used to output the shape of the next block, it will be displayed on the right, beside the grid of the game.

```
270
271     def update_score(nscore):
272         score = max_score()
273
274         with open('scores.txt', 'w') as f:
275             if int(score) > nscore:
276                 f.write(str(score))
277             else:
278                 f.write(str(nscore))
279
280
281     def max_score():
282         with open('scores.txt', 'r') as f:
283             lines = f.readlines()
284             score = lines[0].strip()
285
286         return score
287
```

update_score function

This function is used to change the .txt file, it will change the max score in the .txt file if the current game score is higher than it, and will rewrite the max score if it is still higher than the current score.

max_score function

This function is called at the early start of each game (restarting counts) to get the previously saved max score from the .txt file.

```
def draw_window(surface, grid, score=0, last_score = 0):
    win.blit(bg_img,(0,0))

    pygame.font.init()
    font = pygame.font.SysFont('couriernew', 60, bold = True)
    label = font.render('Tetris', 1, (255, 255, 255))

    surface.blit(label, (top_left_x + play_width / 2 - (label.get_width() / 2), 10))


    font = pygame.font.SysFont('couriernew', 30, bold = True)
    label = font.render('Score: ' + str(score), 1, (255,255,255))

    sx = top_left_x + play_width + 50
    sy = top_left_y + play_height/2 - 100

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


    font = pygame.font.SysFont('couriernew', 25, bold=True)
    label = font.render('High Score: ' + last_score, 1, (255,255,255))

    sx = top_left_x - 250
    sy = top_left_y + 100

    surface.blit(label, (sx + 10, sy + 160))

    # len(grid) is the row on the grid (y-axis), len(grid[i]) is the column (x-axis)
    # extends to 20
    for i in range(len(grid)):
        # extends to 10
        for j in range(len(grid[i])):
            if grid[i][j] in shape_colors:
                win.blit(grid[i][j], (top_left_x + j*block_size, top_left_y + i*block_size))
            else:
                pygame.draw.rect(surface, grid[i][j],
                (top_left_x + j * block_size, top_left_y + i * block_size, block_size, block_size), 0)

    # white rectangle surrounding the grid
    pygame.draw.rect(surface, (255, 255, 255), (top_left_x-10, top_left_y-10, play_width+20, play_height+20), 10, 10)

    draw_grid(surface, grid)
```

draw_window function

Most of the grid UI are put to work here and the main function. This is the function where the grid is finally displayed. This will display the current locations of the blocks onto the grid, whether the grid coordinates contains coloured blocks or simply black. It will also add the border unto the grid UI. This function is called multiple of times later on to keep on updating the current locations of the coloured blocks, in doing so, this function will remove the previous grid and simply redraw the grid to the latest event.

```
425
426 def main_menu(win):
427     run = True
428     pygame.mixer.music.play(-1)
429     while run:
430         win.blit(bg_img,(0,0))
431         draw_text_middle(win, 'Press Any Key To Play', 60, (255,255,255))
432         pygame.display.update()
433         if pygame.mixer.music.get_busy():
434             pass
435         else:
436             pygame.mixer.music.play(-1)
437         for event in pygame.event.get():
438             if event.type == pygame.QUIT:
439                 run = False
440             if event.type == pygame.KEYDOWN:
441                 main(win)
442
443     pygame.display.quit()
```

main_menu function

This function will be the first UI that pops up unto the screen and after losing the game, it will be responsible to restart the game as well as calling the main() function by pressing any key. It also makes it possible to quit the game.

```python
def main(win):
    last_score = max_score()
    locked_positions = {}
    grid = create_grid(locked_positions)

    change_piece = False
    run = True
    current_piece = get_shape()
    next_piece = get_shape()
    clock = pygame.time.Clock()
    fall_time = 0
    fall_speed = 0.27
    level_time = 0
    score = 0

    while run:
        grid = create_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

        if fall_time/1000 > fall_speed:
            fall_time = 0
            current_piece.y += 1
            if not(valid_space(current_piece, grid)) and current_piece.y > 0:
```

```
364                 current_piece.y -= 1
365                 change_piece = True
366 
367         for event in pygame.event.get():
368             keys = pygame.key.get_pressed()
369             if keys[pygame.K_DOWN]:
370                 fall_speed = 0.05
371             else:
372                 fall_speed = 0.27
373             if event.type == pygame.QUIT:
374                 run = False
375                 pygame.display.quit()
376 
377 
378             if event.type == pygame.KEYDOWN:
379                 if event.key == pygame.K_LEFT:
380                     current_piece.x -= 1
381                     if not(valid_space(current_piece, grid)):
382                         current_piece.x += 1
383                 if event.key == pygame.K_RIGHT:
384                     current_piece.x += 1
385                     if not(valid_space(current_piece, grid)):
386                         current_piece.x -= 1
387                 if event.key == pygame.K_DOWN:
388                     current_piece.y += 1
389                     if not(valid_space(current_piece, grid)):
390                         current_piece.y -= 1
391                 if event.key == pygame.K_UP:
392                     current_piece.rotation += 1
393                     if not(valid_space(current_piece, grid)):
```

```
394                     current_piece.rotation -= 1
395
396         shape_pos = convert_shape_format(current_piece)
397
398         for i in range(len(shape_pos)):
399             x, y = shape_pos[i]
400             if y > -1:
401                 grid[y][x] = current_piece.color
402
403         if change_piece:
404             for pos in shape_pos:
405                 p = (pos[0], pos[1])
406                 locked_positions[p] = current_piece.color
407             current_piece = next_piece
408             next_piece = get_shape()
409             change_piece = False
410             score += clear_rows(grid, locked_positions) * 10
411
412         draw_window(win, grid, score, last_score)
413         draw_next_shape(next_piece, win)
414         pygame.display.update()
415
416         if check_lost(locked_positions):
417             draw_text_middle(win, "YOU LOST!", 80, (255,255,255))
418             pygame.display.update()
419             pygame.mixer.music.stop()
420             pygame.mixer.Sound.play(lose_sound)
421             pygame.time.delay(1500)
422             run = False
423             update_score(score)
```

main function

This is the function where all the functions are put to use. The while looping in this main function is what made the objects seems to move possible, as it updates the grid every time it reruns the loop and call upon specific functions.

- *Variable declarations & initialization:* line 335 - 347; These are the variables that are needed to be reset or called each time the main function is called.

- *Time & level:* line 350 - 365; The longer the game goes on, the faster the drop rate of the blocks, thus this is what the increasing level meant. This part of the section will also manage the 'Tetris' blocks to drop down on a certain interval, instead of just standing still on the grid.

- *block movements:* line 367 - 375 + line 378 - 396; the first part of the section determines whether the DOWN arrow key is held down, and will increase the falling speed if it is as well as check for input or changes happening, thus the pygame.event.get() function. The second section of the code (line 378 - 396) will check for the movement of the block (the arrow keys) then move it to the desired position on the grid, however it will check for the valid space before hand if the movement to be made is possible or not, if not it will not change position.

- *change colour:* line 398 - 401; this will change the colours of the blocks that was either translated or rotated from the covert_shape_format() function.

- *locking and changing to new blocks:* line 403 - 410; if the change_piece is True it will lock the last 'Tetris' block and then call for a new shape for the next 'Tetris' block to drop, as well as check if a row is filled with the coloured blocks and whether or not to add the score points.

- *check if lost:* line 416 - 423; this section will check if the 'Tetris' blocks are no longer able to be dropped as the blocks reached the top of the game grid and will break out of the looping.

```
177    def valid_space(shape, grid):
178        accepted_pos = [[(j, i) for j in range(10) if grid[i][j] == (0,0,0)] for i in range(20)]
179        accepted_pos = [j for sub in accepted_pos for j in sub]
180
181        formatted = convert_shape_format(shape)
182
183        for pos in formatted:
184            if pos not in accepted_pos:
185                if pos[1] > -1:
186                    return False
187        return True
```

valid_space function

This function will check if the shape and position on the grid that the 'Tetris' block is set to move there will be possible to move at. It will check whether or not the location on the grid already has a block or is in clash with the border of the grid.

```
190     def check_lost(positions):
191         for pos in positions:
192             x, y = pos
193             if y < 1:
194                 return True
195 
196         return False
```

check_lost function

check_lost will determine if the position of the last ‘Tetris’ block has reached the top of the game grid, by checking on the locked ‘Tetris’ block coordinate on the grid via the dictionary.

```
161 def convert_shape_format(shape):
162     positions = []
163     format = shape.shape[shape.rotation % len(shape.shape)]
164 
165     for i, line in enumerate(format):
166         row = list(line)
167         for j, column in enumerate(row):
168             if column == '0':
169                 positions.append((shape.x + j, shape.y + i))
170 
171     for i, pos in enumerate(positions):
172         positions[i] = (pos[0] - 2, pos[1] - 4)
173 
174     return positions
```

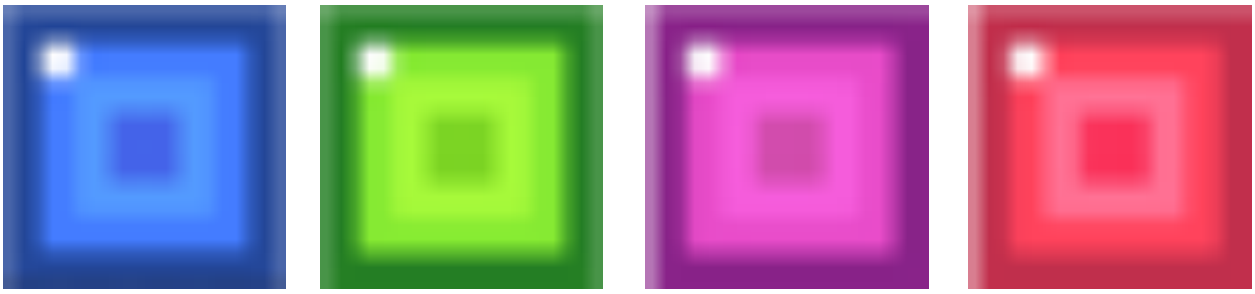
convert_shape_format function

This function will return the new location/coordinate of the ‘Tetris’ block on the grid after being translated/rotated which will then be displayed on the game grid by a different function. This function is what is responsible to getting the new position of the ‘Tetris’ block, after falling, translating, or rotating to be then redisplayed on the game grid.

```
446    win = pygame.display.set_mode((s_width, s_height))
447    pygame.display.set_caption('Tetris')
448    main_menu(win)
```

These last few lines are used to call the main_menu function and to display the window of the game.

Sprites



These images above are the sprites used to represent the colour of each block (30 x 30) of a ‘Tetris’ block.

Background