BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:	**Surname**	**Given Names**	**Student ID Number**
	Sudjoko	**Nicholas Audley**	**2301900321**

Course Code	**: COMP6056**	**Course Name**	**: Program Design Methods and Introduction to Programming**
Class	**: L1AC**	**Name of Lecturer(s)**	**: Ida Bagus Kerthyayana Manuaba**
Major	**: CS**		
Title of Assignment (if any)			
Type of Assignment	**: Final Project**		
Submission Pattern			
Due Date	**: 17-01-20**	**Submission Date**	**: 17-01-20**

The assignment should meet the below requirements.

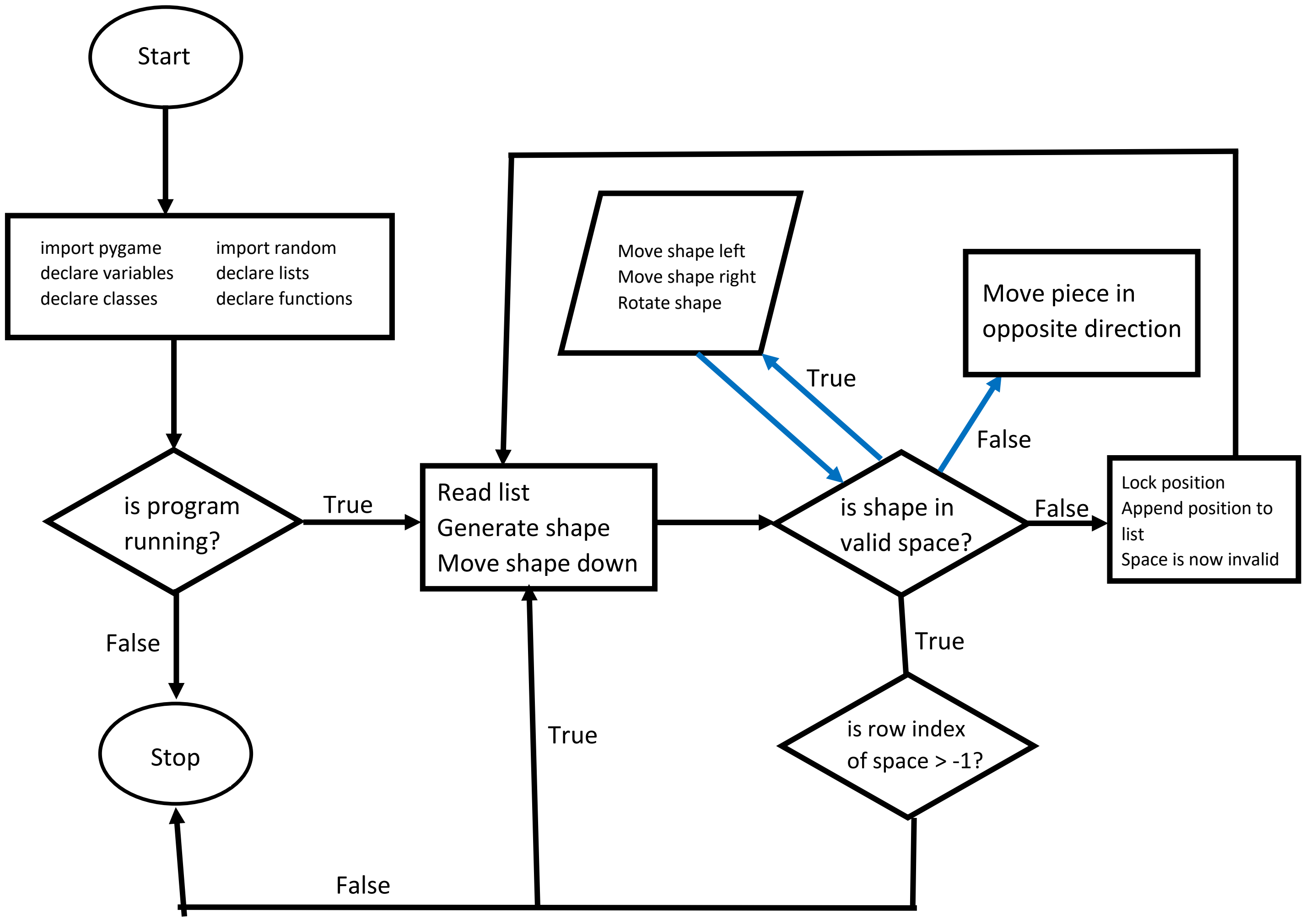
1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.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

I. Project Specifications

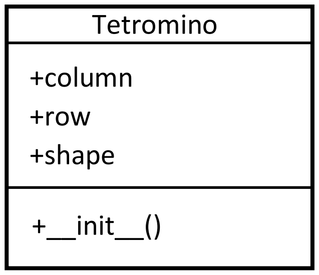
Stack is a program that attempts to replicate the video game Tetris. It is made using Python and the pygame library. The game mechanics for Tetris provides a number of logical problems that needs to be solved in order to make a functioning game. The intention of this project is to provide myself a learning experience on the field of video game development. This project can be considered simple as it only uses one module. However, there is one complication. Stack fails to replicate one mechanic from Tetris, which is the mechanic that allows the game to empty rows that are completely filled with blocks. This mechanic is relatively complicated to code with my current level of experience.

II. Solution Design

Flowchart



Class Diagram



III. Discussion

There are many comments already in the code but we will go through them here
The first thing we need to do is of course import 'pygame' and 'random' libraries
Then we initialize the pygame.font to allow rendering of fonts for text

```python
import pygame
import random

pygame.font.init()
```

Next we declare all the variables we need to start

```python
# Variables for Game Area Size
game_width = 200
game_height = 400

# Variable for size of a "square" in the Game Area
square_size = 20

# Variables for Window Size
window_width = 500
window_height = 550

# The left side of the game area in x-axis
gridOrigin_x = (window_width // 2) - (game_width // 2)

# The top of the game area in y-axis
gridOrigin_y = window_height - game_height
```

We then create a number of multidimensional lists to hold the shape of pieces used in the game
For the piece "I" there are two lists because when you rotate the shape there is only 2 possible states

```python
I = [['..x..',
      '..x..',
      '..x..',
      '..x..',
      '.....'],
     ['.....',
      'xxxx.',
      '.....',
      '.....',
      '.....']]
```

We want to add color to the pieces later so we make 2 more lists to hold and assign them

```
# List of Shapes
shapes = [O, I, S, Z, L, J, T]

# List of colors for the Shapes
shape_colors = [(255, 213, 0), (66, 165, 245), (114, 203, 59), (255, 50, 19),
                (3, 65, 174), (255, 151, 28), (191, 0, 250)]
```

Now we make a class here to hold the properties of the pieces

```
class Tetromino(object):
    def __init__(self, column, row, shape):
        self.column = column
        self.row = row
        self.shape = shape
        self.color = shape_colors[shapes.index(shape)]
        self.rotate = 0
```

Then for our first function we loop the color black through 10 columns and 20 rows to create a grid
The locked_pos parameter is there so we can "lock" a piece/shape in the game later

```
# This function establishes a 10 by 20 grid for the game to take place
def game_grid(locked_pos={}):  # We make a dictionary here to contain position and color of a
locked shape
    # We make a grid by looping
    grid = [[(0, 0, 0) for _ in range(20)] for _ in range(10)]

    for column in range(len(grid)):
        for row in range(len(grid[column])):
            if (column, row) in locked_pos:
                lock = locked_pos[(column, row)]
                grid[column][row] = lock
    return grid
```

The next function is pretty self-explanatory, it draws the lines so we can actually see the grid
pygame.draw.line() takes 4 arguments, surface, color, start position and end position

```
# This function draws the lines for the grid
def draw_grid(surface, grid):
    x = gridOrigin_x
    y = gridOrigin_y

    for column in range(len(grid)):
        # Draws Vertical Lines
        pygame.draw.line(surface, (255, 255, 255), (x + column*square_size, y),
                                 (x + column*square_size, y + game_height))
        for row in range(len(grid[column])):
            # Draws Horizontal Lines
            pygame.draw.line(surface, (255, 255, 255), (x, y + row*square_size),
                                 (x + game_width, y + row*square_size))
```

The next function is needed if you want to see the game at all. We fill the window with the color black, insert title text, color in the grid, draw the grid lines, and draw a border around the grid.

```
# This is the function that lets us see everything we just made
def draw_window(surface, grid):
    # Fills the window with a color, this is our surface, we can change the color by changing
rgb values
    surface.fill((0, 0, 0))

    # This is the title text above the game, we can change font type, size, color, position
    pygame.font.init()
    font = pygame.font.SysFont('helvetica', 40)
    label = font.render('Stack the Blocks', 1, (255, 255, 255))
    surface.blit(label, (gridOrigin_x + game_width // 2 - (label.get_width() // 2), 75))

    # This fills the squares in the grid with colors
    for column in range(len(grid)):
        for row in range(len(grid[column])):
            pygame.draw.rect(surface, grid[column][row], (gridOrigin_y + column*square_size,
                             gridOrigin_x + row*square_size, square_size, square_size), 0)

    # We call draw_grid() here so we can actually see the grid
    draw_grid(surface, grid)

    # We use pygame.draw.rect() to create a border around our grid, so we know the physical
borders for the game
    pygame.draw.rect(surface, (255, 255, 255), (gridOrigin_x, gridOrigin_y, game_width,
game_height), 5)
    pygame.display.update()
```

Short function here that uses the random module to generate a random piece when the game runs

```
# This function generates a random piece when the game runs
def get_piece():
    return Tetromino(5, 0, random.choice(shapes))
```

We've seen the multidimensional list at the start, and while we can see the shape, it doesn't make sense to the computer. What we need to do now is make a blank list that can hold the positions of the letter 'x' which represents which square needs to be filled with a color to make the piece.

We make a loop that goes through the rows and columns to check for the position of 'x', and when we find 'x' we add the position (which is the index numbers of row and columns) into the list.

```
# This function makes sense of the list of shapes earlier
def draw_piece(shape):
    # We make a blank list called positions
    positions = []
    # Now we decide which orientation of shape we will draw, by default the first index is 0 so
its the first one
    drawn = shape.shape[shape.rotate % len(shape.shape)]

    # Checks through the columns and rows for 'x', enumerate() takes the position of x
    for x, vertical in enumerate(drawn):
        column = list(vertical)
        for y, row in enumerate(column):
            if row == 'x':
                positions.append((shape.column + x, shape.row + y))  # Then adds the position
of 'x' to list if found
```

```
                # We can see this list with print(positions)

    for x, position in enumerate(positions):
        positions[x] = (position[0] - 2, position[1] - 4)  # - to ignore the '.' in the list
which changes the position
    return positions
```

The next function is important, we don’t want the pieces to move through each other, so we need to define which space that pieces can move in, and which space they cannot move in.

We make another list, and it holds the positions of all the squares that are the color black (black = empty).

```
# This function determines the spaces that are valid for pieces to move in
def valid_space(shape, grid):
    # Makes a list of valid positions, if the position is filled (not color black) it is
counted as invalid
    accepted_positions = [[(column, row) for row in range(20) if grid[column][row] == (0, 0,
0)]
                          for column in range(10)]
    accepted_positions = [row for _ in accepted_positions for row in _]  # Convert embedded
list to one list

    drawn = draw_piece(shape)

    for position in drawn:
        if position not in accepted_positions:
            if position[1] > -1:  # position[1] is the y-coordinate, when >= 0 means it is in
the grid and valid space
                return False
    return True
```

This is another short one. In the game of Tetris if your piece reaches the top of the grid you lose, so what we did here is make a function that checks if the position of the piece is above the grid.

```
# This function is called if a piece is locked at a y value of <= 1 (above the grid)
def lost_condition(positions):
    for position in positions:
        x, y = position
        if y < 1:
            return True
    return False
```

The next function is the longest one. First, we create the mechanic that allows the pieces to fall down the grid. There are variables here that we can change the value of to change the speed of the fall.
Making the pieces fall means that it might go through other pieces, to prevent this we call valid_space() from earlier.

```
# These are the base game mechanics
def game():
    locked_pos = {}

    run = True
    change_piece = False
    current_piece = get_piece()
    next_piece = get_piece()
    game_clock = pygame.time.Clock()  # creates an object to help track time
    falling_speed = 0
    falling_time = 0.3  # increasing this value makes the pieces fall slower

    while run:
        grid = game_grid(locked_pos)
        falling_speed += game_clock.get_rawtime()  # get_rawtime() counts the time until tick()
        game_clock.tick()

        if falling_speed/1000 > falling_time:
            falling_speed = 0
            current_piece.row += 1
            if not (valid_space(current_piece, grid)) and current_piece.row > 0:
                current_piece.row -= 1
                change_piece = True  # Stops moving the current piece, locks it, and generates 
the next piece
```

Second, we create the buttons that can control movement and rotation of the piece

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

    if event.type == pygame.KEYDOWN:

        # Moves the piece to the left by 1 column
        if event.key == pygame.K_LEFT:
            current_piece.column -= 1
            if not valid_space(current_piece, grid):
                current_piece.column += 1

        # Moves the piece to the right by 1 column
        if event.key == pygame.K_RIGHT:
            current_piece.column += 1
            if not valid_space(current_piece, grid):
                current_piece.column -= 1

        # Moves the piece down by 1 row
        if event.key == pygame.K_DOWN:
            current_piece.row += 1
            if not valid_space(current_piece, grid):
                current_piece.row -= 1

        # Rotates the piece
        if event.key == pygame.K_UP:
            current_piece.rotate += 1
            if not valid_space(current_piece, grid):
                current_piece.rotate -= 1

shape_position = draw_piece(current_piece)
```

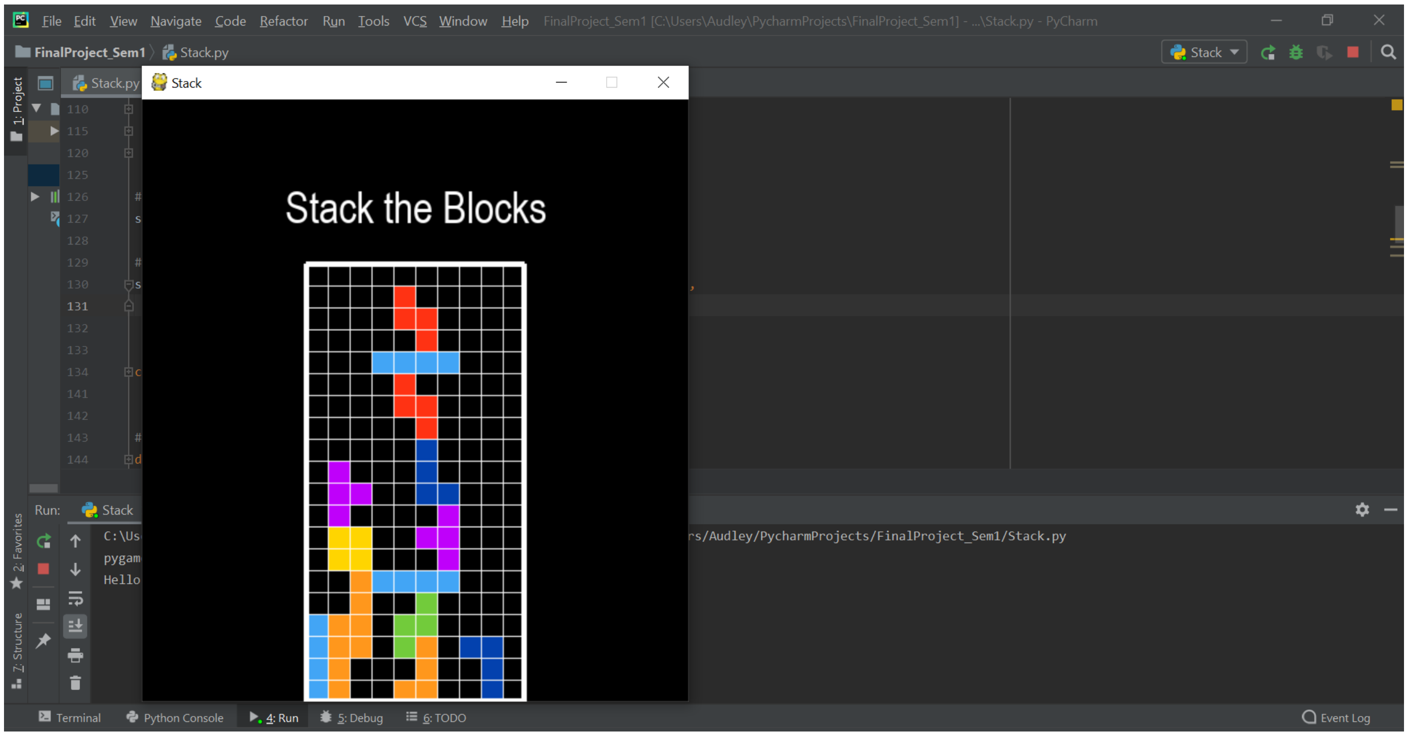
Next we allow change_piece to generate a new piece once the current one has stopped moving.
In the place of the piece that just stopped moving we fill the grid with the colors of the piece.
Lastly we make this a recursive function, when the lost condition is met the function game() calls itself to start a fresh game.

```
shape_position = draw_piece(current_piece)

    for y in range(len(shape_position)):
        row, column = shape_position[y]
        if column > -1:
            grid[row][column] = current_piece.color  # Draws the color of the piece

    if change_piece:
        for position in shape_position:
            pos = (position[0], position[1])
            locked_pos[pos] = current_piece.color
        current_piece = next_piece
        next_piece = get_piece()
        change_piece = False

    if lost_condition(locked_pos):
        game()  # Starts new game immediately if lost condition is met

    draw_window(surface, grid)

pygame.display.quit()
```

Now we initialize the video system so we can see everything

```
# Initializes video system
surface = pygame.display.set_mode((window_width, window_height))
pygame.display.set_caption('Stack')
game()
```

IV. Screenshot


File Edit View Navigate Code Refactor Run Tools VCS Window Help FinalProject_Sem1 [C:\Users\Audley\PycharmProjects\FinalProject_Sem1] - ...\Stack.py - PyCharm
FinalProject_Sem1
Stack.py
Stack
Stack the Blocks
Run:
Terminal
Python Console
4: Run
5: Debug
6: TODO
Event Log

```
C:\Us                                                    rs/Audley/PycharmProjects/FinalProject_Sem1/Stack.py
pygam
Hello
```