

Sunday, 15th of January 2023

Documentation for Algorithm & Programming Final Project

“Tiny Tetris Bot”



Lecturer:

Jude Joseph Lamug Martinez, MCS

Report done by:

Rafael Sutiono - 2602174535

**BINUS University International
2023**

Table of Contents

A. Description	3
B. Diagrams	4
1. Use-case Diagram	4
2. Activity Diagram	5
3. Class Diagram	5
C. Modules	6
D. Classes	7
1. TetrisPieces	7
E. Essential Algorithms	8
1. Displaying the Board	8
2. Random Piece Generator	9
3. Rotations	10
4. Wall Kicks	11
5. Clearing Full Lines	13
6. User Input	14
7. Game Over	15
8. Running the Game Loop	16
F. Screenshots	17
G. Lessons Learnt	20

A. Description

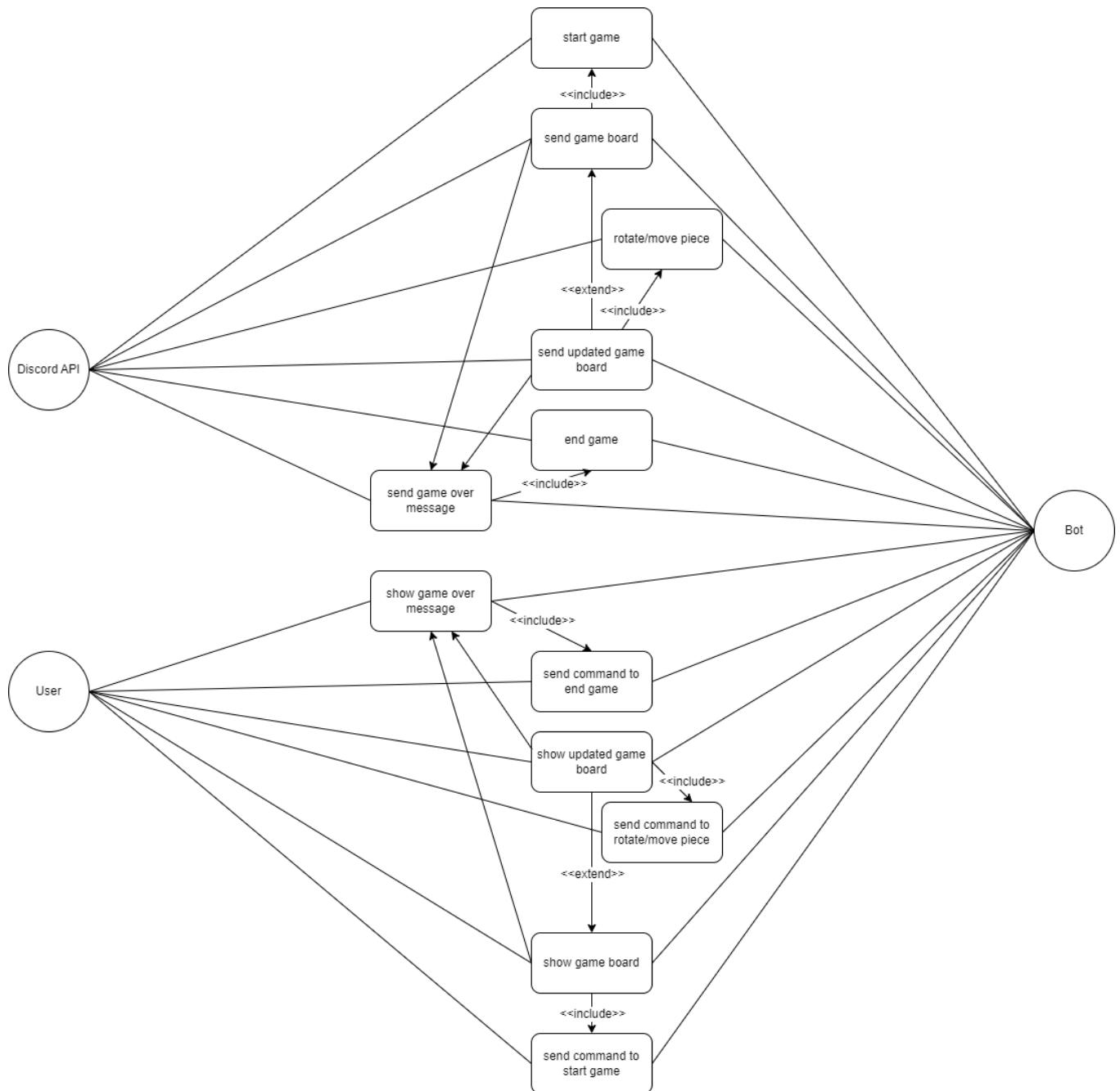
The Tiny Tetris Bot allows you to play the classic game of Tetris right within your Discord server. With a variety of Tetris pieces to play with, you can challenge your friends and compete for the highest score. The bot also features a variety of game controls, such as the ability to rotate and move pieces, as well as the ability to clear lines. Additionally, the bot tracks your score and number of lines cleared, giving you a sense of progression as you play.

The game is played in an interactive way, by using discord reactions as a control method. The bot also features an embedded message that updates in real-time, giving you a clear view of the game board and your progress. The game board is designed to be immersive, with different colored squares representing the different Tetris pieces. The bot is also designed to be user-friendly, with easy-to-use controls and a simple interface. The game also includes a game over feature and allows to restart the game again with just one press of a button. Unfortunately, due to Discord's API rate limit, the game progresses very slowly—this may change in the future depending on Discord's end.

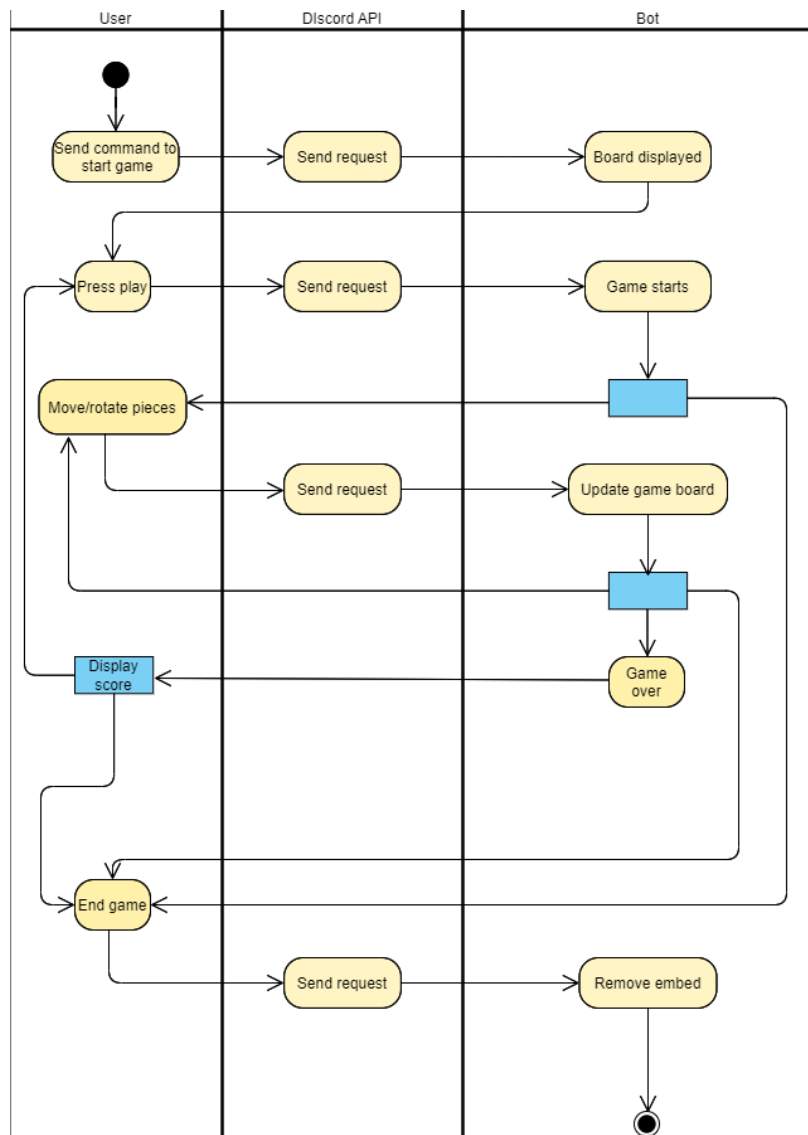
The code for this bot is written in Python, using the discord.py library and the asyncio library for running coroutines. It also uses RNG to make the game more interesting and challenging. Additionally, the code utilizes the dotenv library to store the bot's token, making it easy to run the bot on your own server.

B. Diagrams

1. Use-case Diagram

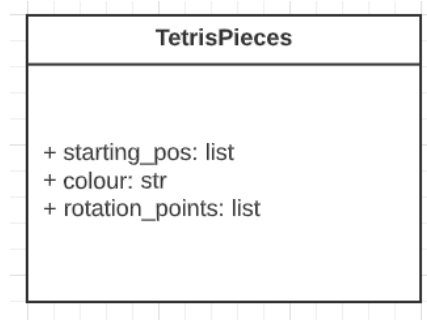


2. Activity Diagram



3. Class Diagram

Only one class is initialized in the code:



C. Modules

The following table lists down and briefly describes the modules used in creating the Tiny Tetris Bot.

Module	Description
discord	This module is used to interact with the Discord API and create a bot client. It is used to handle events and send messages to a Discord server.
discord.ext.commands	This module is used to create commands for the bot, which can be invoked by users in a Discord server.
random	This module is used to generate random numbers, which are used to choose the next shape in the game.
asyncio	This module is used to run coroutines, which are lines of code that act on the occurrence of an event loop.
os	This module is used to interact with the operating system, in this case to retrieve the bot's token from the environment variable.
dotenv	This module is used to load environment variables from a .env file. It is used to get the bot's token.

D. Classes

1. TetrisPieces

The TetrisPieces class is used to represent the different Tetris pieces that can appear in the game. It is defined with three attributes: *starting_pos*, *colour*, and *rotation_points*.

- The *starting_pos* attribute is a list that represents the starting position of the piece on the game board.
- The *colour* attribute is a string that represents the colour of the piece, using emoji strings in discord.
- The *rotation_points* attribute is a list of lists that stores data for rotating the Tetris piece.

The class is instantiated when a new piece is created and its attributes are set based on the arguments passed to the constructor. The class does not have any methods, but it is used in the *rotate_shape* function to determine the new position of the shape after rotation. By providing the data for the rotation of each piece, the class allows for the smooth rotation of each piece in the game.

E. Essential Algorithms

1. Displaying the Board

```
def format_board_as_str():  
    # create an empty string that will hold the  
    board_as_str = ''  
    for row in range(no_of_rows):  
        for col in range(no_of_cols):  
            # adds emojis for each square to th  
            board_as_str += (board[row][col])  
            # Add a newline character after the  
            if col == no_of_cols - 1:  
                board_as_str += "\n "  
    # return the string representation of the g  
    return board_as_str
```

snippet of the `format_board_as_str` function

The `format_board_as_str` function is responsible for formatting the current state of the game board as a string so that it can be displayed as the message content in Discord. This function is crucial because it allows the game board to be easily displayed to the user and also allows for easy updates to the board as the game is played.

The function iterates through the 2D list called `board` and adds the corresponding emoji string to a new string called `board_str` for each element in the list. Once all elements have been added, the function returns the `board_str` which can then be displayed in the Discord message.

2. Random Piece Generator

```
def get_random_shape():
    global index
    shapes = [shape_I, shape_J, shape_L, shape_O, shape_S, shape_T, shape_Z]
    # set random_shape as a random shape object from the above list, which is done by generating a random integer between 0 and 6.
    random_shape = shapes[random.randint(0, 6)]
    index += 1
    # if start_higher is True, the function iterates over the starting_pos attribute of the random_shape object and subtracts 1 from the row value of each square in the attribute
    # this effectively moves the tetris piece up one row on the game board
    if start_higher == True:
        for s in random_shape.starting_pos[:]:
            s[0] = s[0] - 1
    else:
        # if start_higher is False, the function initializes a variable called starting_pos and sets it equal to a copy of the starting_pos attribute of the random_shape object.
        starting_pos = random_shape.starting_pos[:]
    random_shape = [random_shape.starting_pos[:], random_shape.colour, random_shape.rotation_points] #gets starting point of shapes and copies, doesn't change them
    global is_new_shape
    is_new_shape = True
    # returns random_shape as a list containing starting_pos, colour, and rotation_points attributes
    # effectively allowing the game to track the position, colour, and rotation data of the randomly generated tetris piece.
    return random_shape
```

snippet of the get_random_shape function

The `get_random_shape` function is responsible for randomly selecting and returning a Tetris piece to be placed on the game board. It does this by generating a random number between 0 and 6, and using that number to select a specific piece from a list of pre-defined Tetris pieces. Each piece is represented by a list of lists, where each inner list contains the coordinates of a square that makes up the piece. The color of the piece is also included.

The function is important as it determines the sequence of Tetris pieces that will appear in the game, adding to the randomness and replayability of the game.

3. Rotations

```
def rotate_shape(shape, direction, rotation_point_index, shape_colour):
    rotation_point = shape[rotation_point_index] # assigns the value of the square at the rotation_point_index of the shape variable to the rotation_point variable
    new_shape = [] # to store coords of rotated shape

    for square in shape:
        # assign the value of the first and second element of the square to square_row and square_col respectively
        square_row = square[0]
        square_col = square[1]
        # check if direction is clockwise
        if direction == 'clockwise':
            # calculate the new positions of the square after a clockwise rotation
            new_square_row = (square_col - rotation_point[1]) + rotation_point[0] + rot_adjustments.get(shape_colour)[rotation_pos-1][0]
            print('Adjustment made: ' + str(rot_adjustments.get(shape_colour)[rotation_pos-1][0]))
            new_square_col = -(square_row - rotation_point[0]) + rotation_point[1] + rot_adjustments.get(shape_colour)[rotation_pos-1][1]
            print('Adjustment made: ' + str(rot_adjustments.get(shape_colour)[rotation_pos-1][1]))
            new_shape.append([new_square_row, new_square_col]) # store position of rotated square
        if (0 <= square_col < no_of_cols) and (0 <= square_row < no_of_rows): # check if the new square is within the game board
            board[square_row][square_col] = empty_sq # make the square at the old position empty on the game board
```

part of the rotate_shape function

The *rotate_shape* function is responsible for rotating the current Tetris piece being played. It takes in three parameters: *shape*, *direction*, and *rotation_point_index*. The *shape* parameter is the current Tetris piece's position on the game board, the *direction* parameter is either "clockwise" or "counter-clockwise" indicating the direction of rotation, and the *rotation_point_index* parameter is the index of the square in the *shape* list that serves as the rotation point.

The function first assigns the value of the square at the *rotation_point_index* of the *shape* variable to the *rotation_point* variable and then creates an empty list *new_shape* to store the coordinates of the rotated shape. It then iterates through each square in the *shape* variable and calculates the new positions of the square after a clockwise rotation. The new positions are then stored in the *new_shape* list. The function then returns the new shape.

The importance of this function is that it enables the player to rotate the Tetris piece in the game, making it more challenging and fun to play. Rotating the Tetris piece allows the player to fit it into tight spaces and clear more lines.

4. Wall Kicks

```
def do_wall_kicks(shape, old_shape_pos, shape_colour, attempt_kick_num):
    new_shape_pos = []

    # determine which set of wall kick to use based on shape colour
    if shape_colour == blue_sq:
        kick_set = main_wall_kicks[rotation_pos]
    else:
        kick_set = i_wall_kicks[rotation_pos]

    print('Kick set: ' + str(kick_set))
    for kick in kick_set:
        print('Kick: ' + str(kick))
        for square in shape:
            # assign the value of the first and second element of the square
            square_row = square[0]
            square_col = square[1]
            # assign the value of square_row plus the value of the first element of the kick
            new_square_row = square_row + kick[0]
            new_square_col = square_col + kick[1]
```

small section of the do_wall_kicks function

The *do_wall_kicks* function is used to handle the adjustments made to the position of a Tetris piece after it has been rotated. The purpose of this function is to ensure that a Tetris piece doesn't get stuck against the wall or another piece after it has been rotated. This function takes in four parameters: *shape*, *old_shape_pos*, *shape_colour*, and *attempt_kick_num*.

The variable *shape* represents the current shape of the Tetris piece, *old_shape_pos* represents the previous position of the Tetris piece, *shape_colour*

represents the color of the Tetris piece, and *attempt_kick_num* represents the number of times the code has attempted to rotate the shape.

The function first checks which set of wall kicks to use based on the *shape_colour*. If the *shape_colour* is *blue_sq*, the function uses the *main_wall_kicks* set, otherwise, it uses the *i_wall_kicks* set. The function then iterates through each square in the shape and assigns the value of the first and second elements of the square to *square_row* and *square_col* respectively. It then assigns the value of *square_row* plus the value of the first element of kick to *new_square_row* and *square_col* plus the value of the second element of kick to *new_square_col*.

The function then checks if the new square is within the game board and checks the square to check if it is empty. If it is not empty or if the new position of the square is not in the *old_shape_pos*, it means the shape doesn't fit, so the *new_shape_pos* is reset and the loop breaks. However, if the square is empty and the new position is in the *old_shape_pos*, it means the shape fits, and so the new position of the square is appended to the *new_shape_pos* list.

If the *new_shape_pos* list reaches 4, it means the shape has been successfully rotated and the *new_shape_pos* is returned. If the *new_shape_pos* list does not reach 4, it means the shape could not be rotated and so the old, unrotated shape is returned. This function is important as it ensures that the shape can rotate without getting stuck on the wall or another shape, allowing for a smooth gameplay experience.

5. Clearing Full Lines

```
def clear_lines():
    # make the board, points and lines variables global,
    global board
    global points
    global lines
    lines_to_clear = 0
    for row in range(no_of_rows):
        row_full = True # assume line is full
        for col in range(no_of_cols):
            if board[row][col] == empty_sq:
                row_full = False
                break # exit nested loop if the current s
        if row_full:
            lines_to_clear += 1
            board2 = board[:] # create a copy of the board
            for r in range(row, 0, -1): # nested loop that
                if r == 0: # check if current row is the
                    for c in range(no_of_cols): # nested
                        board2[r][c] = empty_sq # make the
                else:
                    for c in range(no_of_cols):
                        board2[r][c] = board[r - 1][c] #
            board = board2[:] # make the board variable e
```

small section of the clear_lines function

The `clear_lines` function is used to check for and clear any full lines on the game board. The function iterates through each row of the game board and checks if every square in the row is filled with a colored square. If it is, the points variable is incremented by 1 and the lines variable is incremented by 1. The full row is then cleared by replacing each square with the empty square emoji. This function is vital as it allows the player to gain points and clear lines in order to progress through the game. It also keeps the game board from becoming too cluttered, making it easier for the player to continue playing the game.

The `clear_lines` function is called each time a Tetris piece is placed on the board, ensuring that any full lines are cleared before the next piece is placed.

6. User Input

```
if str(reaction.emoji) == "←": # Left button pressed
    print('Left button pressed')
    h_movement = -1 # move 1 left
    await msg.remove_reaction("←", user)
if str(reaction.emoji) == "→": # Right button pressed
    print('Right button pressed')
    h_movement = 1 # move 1 right
    await msg.remove_reaction("→", user)
if str(reaction.emoji) == "↓": # Down button pressed
    print('Down button pressed')
    global down_pressed
    down_pressed = True
    await msg.remove_reaction("↓", user)
if str(reaction.emoji) == "🕒": # Rotate button pressed
    print('Rotate clockwise button pressed')
    global rotate_clockwise
    rotate_clockwise = True
    if rotation_pos < 3:
        rotation_pos += 1
    else:
        rotation_pos = 0 # go back to original position
    await msg.remove_reaction("🕒", user)
if str(reaction.emoji) == "✖": # Stop game button pressed
    await reset_game()
    await msg.delete()
if str(reaction.emoji) == "🔴":
    await msg.edit(content="")
```

part of the `on_reaction_add` function

The algorithm for handling user input in the code is based on the use of the `discord.py` library, which allows the bot to interact with the Discord API. The code uses the `on_raw_reaction_add` and `on_raw_reaction_remove` events to detect when

a user adds or removes a reaction to the bot's message and then updates the game state accordingly. The code uses a series of global variables, such as *down_pressed*, *rotate_clockwise*, and *h_movement*, to track user input and update the game state.

The importance of this algorithm in the code is that it allows the bot to respond to user input in real time, which is essential for the proper functioning of the Tetris game. Without the ability to detect and respond to user input, the game would not be able to move the Tetris pieces or rotate them, and the game would not be interactive.

7. Game Over

```
if not game_over:
    # update board
    embed = discord.Embed(description=format_board_as_str(), color=embed_colour)
    h_movement = 0 # reset horizontal movement
    rotate_clockwise = False # reset clockwise rotation
    await msg.edit(embed=embed)
    if not is_new_shape:
        await asyncio.sleep(1) # to keep under discord's API rate limit
    await run_game(msg, cur_shape)
else:
    print('GAME OVER')
    desc = 'Score: {} \n Lines: {} \n \n Press ► to play again.'.format(points, lines)
    embed = discord.Embed(title='GAME OVER', description=desc, color=embed_colour)
    await msg.edit(embed=embed)
    await msg.remove_reaction("←", client.user) # Left
    await msg.remove_reaction("↓", client.user) # Down
    await msg.remove_reaction("→", client.user) # Right
    await msg.remove_reaction("🌀", client.user) # Rotate
    await msg.add_reaction("►") # Play
```

snippet of the algorithm that checks if the game is over

The game-over algorithm is used to determine when the game of Tetris has ended; it is part of the *run_game* function that keeps the game running. The algorithm checks if the game is over by checking the variable *game_over*. If the

variable is set to true, the game ends and the game-over message is displayed to the user. The message displays the user's final score and the number of lines cleared. The algorithm also includes the option to play again by pressing the play button.

The game-over algorithm is important as it ensures that the game ends when the game is over and notifies the user when the game has ended, and so giving the user the option to start over.

8. Running the Game Loop

```
async def run_game(msg, cur_shape):
    global is_new_shape
    global h_movement
    global rotate_clockwise
    global rotation_pos

    cur_shape_pos = cur_shape[0] # assign the value of the first element of the cur_shape list to the variable
    cur_shape_colour = cur_shape[1] # assign the value of the second element of the cur_shape list to the varia

    # checks if the variable rotate_clockwise is equal to True and the variable cur_shape_colour is not equal t
    if rotate_clockwise == True and cur_shape_colour != yellow_sq:
        cur_shape_pos = rotate_shape(cur_shape_pos, 'clockwise', cur_shape[2][rotation_pos], cur_shape_colour)
        cur_shape = [cur_shape_pos, cur_shape_colour, cur_shape[2]] # update shape

    next_pos = get_next_pos(cur_shape_pos)[: ]
    movement_amnt = next_pos[0]
    next_space_free = next_pos[1]
```

small section of the run_game function

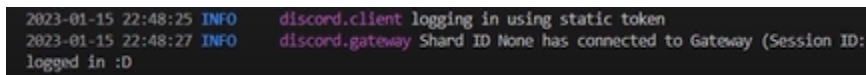
The *run_game* function is the heart of the Tetris bot. It is an asynchronous function that is repeatedly called to update the game state and display the updated game board to the user.

The function takes two parameters, *msg* and *cur_shape*, where *msg* is the message object representing the current game board and *cur_shape* is the current Tetris shape that is being played. The function starts by initializing some global

variables and then uses these variables to determine the next position of the current shape. The next position is determined by checking if the shape can move down one row and if the next position is available. If the next position is available, the function updates the game board by replacing the old position of the shape with an empty square, and the new position with the current shape. If the next position is not available, the function clears any lines that are full and gets a new random shape. This process repeats until the game is over.

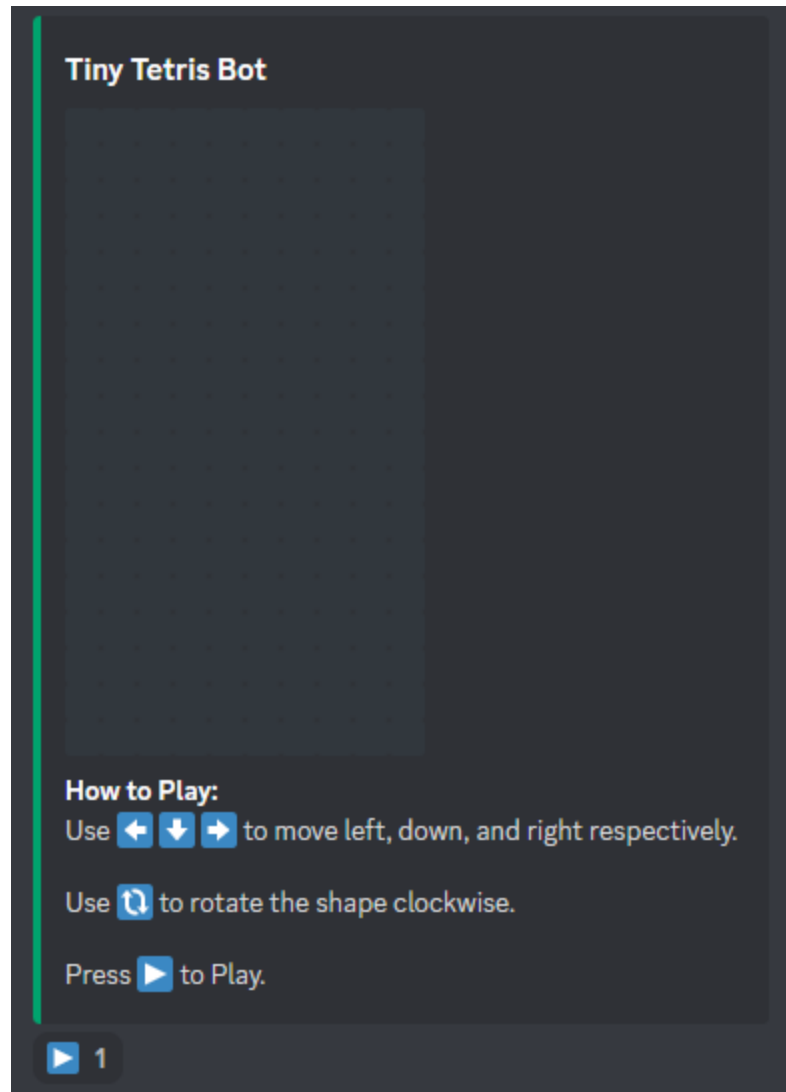
The *run_game* function is crucial because it is responsible for updating the game board, handling user input, and checking for [game-over conditions](#). It is the core of the game logic and without it, the game would not function.

F. Screenshots

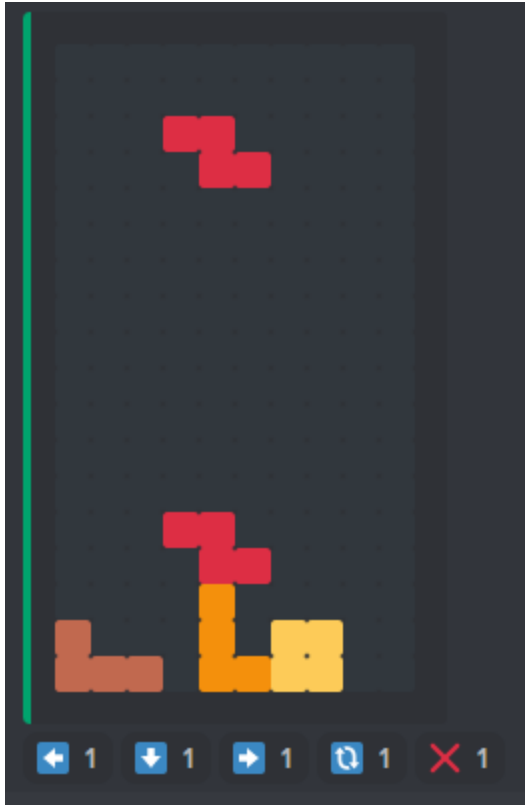


```
2023-01-15 22:48:25 INFO discord.client logging in using static token
2023-01-15 22:48:27 INFO discord.gateway Shard ID None has connected to Gateway (Session ID:
logged in :D
```

logged in :D



instructions

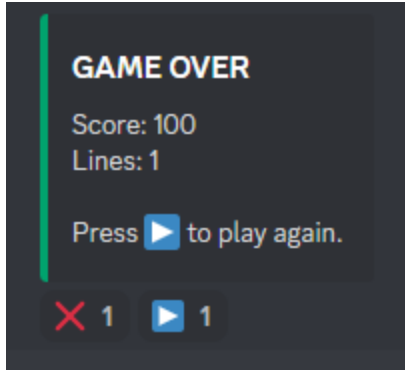


running the game (and throwing it)

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Adjustment made: 0
Adjustment made: 0
Adjustment made: 0
Adjustment made: 0
Kick set: [[0, 0], [0, -1], [0, 2], [-2, -1], [1, 2]]
Kick: [0, 0]
New shape: [[3, 5]]
New shape: [[3, 5], [4, 5]]
New shape: [[3, 5], [4, 5], [5, 5]]
New shape: [[3, 5], [4, 5], [5, 5], [4, 6]]
Returned new shape after doing kicks
Rotated shape: [[5, 5], [4, 5], [4, 6], [3, 5]]
Left button pressed
Down button pressed
Detected a space that isnt free
```

making sure everything works by printing every input



game over

G. Lessons Learnt

I've learnt that it's essential to double-check your work while coding, as the smallest mistake can cost you a lot of time trying to locate it. As such, most of the time spent while coding is focused on fixing errors in your code, though in this process I've learnt so much about debugging. While those are some of the frustrating parts of creating this project, there are some positive aspects I've grasped as well. I've learnt how to run a Discord bot in python and make use of the discord.py library, and at the same time learn so much more about asynchronous lines of code. I've also come to understand that commenting on your code makes it so that you don't forget sections of your code and helps users understand your code as well.

Time management is also an essential part of the process. Not procrastinating on work saves you a lot of hassle, especially if it comes to final projects. Thankfully, I did not spend the entirety of my Christmas Break being unproductive and worked on simultaneous final projects.

To wrap this report up, I would like to say it has been a pleasure working on this project. Not only have I learnt many more aspects of coding that weren't taught in class, but I've also learnt some life lessons along the way.