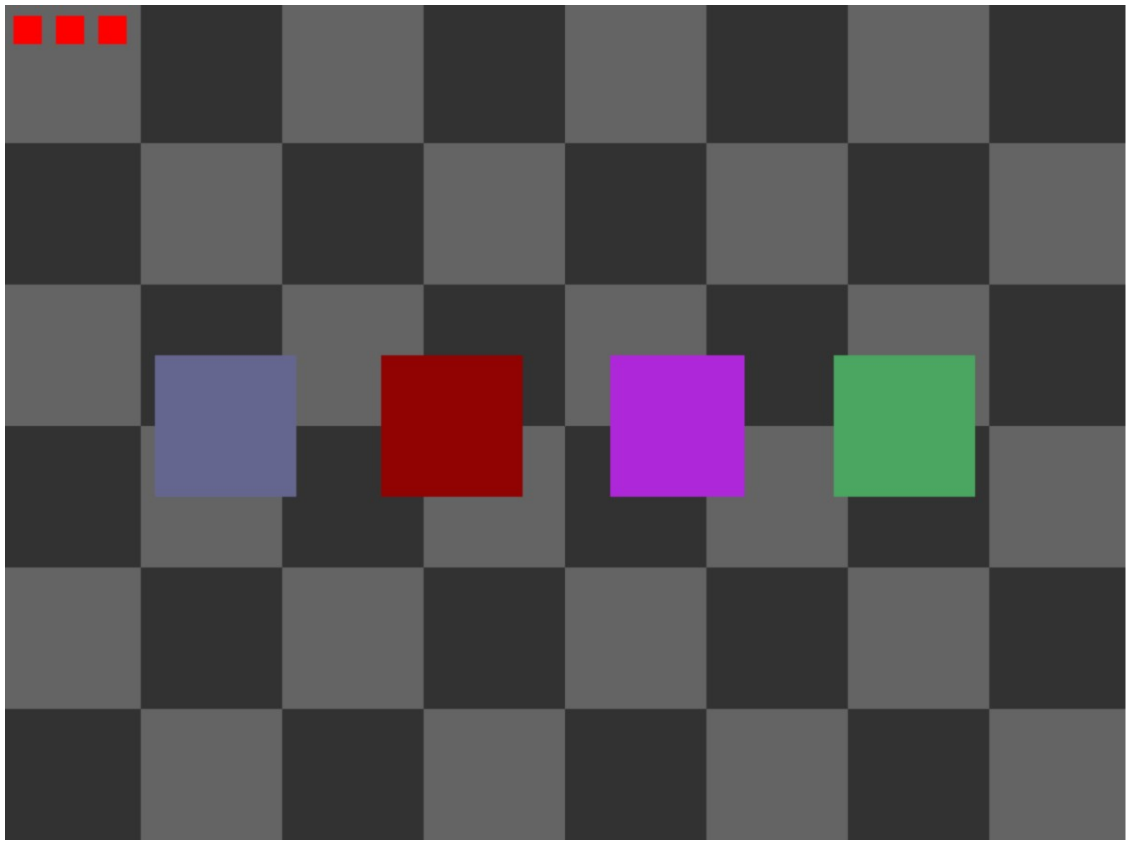
Squares All Round: A Pygame Game

Squares All Round is an engaging pixelated game that challenges players to identify the odd square among a vibrant collection of four squares. The game uses the Pygame library to create a graphical user interface and handle user input.

Game Mechanics

The game displays four squares with randomly generated colors. One of the squares is slightly different from the others - it has a different width. The player has to click on the square that they think is different. If they guess correctly, they gain a point and the screen flashes green. If they guess incorrectly, they lose a life and the screen flashes red.

The player starts with three lives, represented by heart symbols at the top of the screen. If the player loses all their lives, the game ends.



Handling User Input

The game uses Pygame's event handling system to detect when the player clicks on one of the squares. When the player clicks on a square, the `check_guess` function is called to determine if they clicked on the correct square. If the player guesses wrong, this function updates the remaining lives accordingly.

Code Structure

The code is structured into several functions that handle different aspects of the game mechanics:

- `draw_random_colors`: Generates random colors for the squares.
- `check_guess`: Checks if the player clicked on the correct square.
- `game_over`: Displays the game over screen with the final score.
- `generate_x_y`: Calculates the position of squares on screen.

The game generates random colors for the squares using the `draw_random_colors` function. This function uses the `random.randint` function to generate random values for the red, green, and blue components of each color. It also generates a random width for one of the squares - this is the square that will be different from the others.

The main game loop makes sure that the game is run continously until a specific condition is met (e.g. the player loses). It handles user input and updates the game graphics on the screen.

Running The Code

To run this code, you will need to have Python and a user interface (UI) for it installed on your computer with pygame in your UI's library. You can then run this code by opening a terminal or command prompt and pasting it there.

```python
import pygame
import random

#### Start pygame
pygame.init()

#### Setting up game window
screen_width, screen_height = 800, 600
screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption("Squares all round")

#### Global variables
colors = [] # List of colors to e added
target_index = 0 # The target square that will differ in width
lives = 3
score = 0

#### Checkerboard pattern configuration
tile_size = 100
square_size = 100
square_distance = 60 # Distance between squares
```

```
#### Generates four random colors with varying width
def draw_random_colors():

    global colors, target_index

    colors = []
    target_index = random.randint(0, 3) # Picking which square will be
the odd one (changes per round)

    for i in range(4): # NOTE to self: 0-3, 0 incl
        red = random.randint(0, 255)
        green = random.randint(0, 255)
        blue = random.randint(0, 255)

        if i == target_index:
            width = random.randint(90, 110)
            colors.append((red, green, blue, width))
        else:
            width = 100 # random.randint(40, 80) # OBS! If changed,
make sure to not have an overlap with the target_index
            colors.append((red, green, blue, width))

def check_guess(clicked_index):
    global lives, score

    if clicked_index == target_index:
        print("Congratulations! You guessed correctly")
        score += 1
        screen.fill((0, 255, 0))  # Set screen color to green
    else:
        print("Oops! That's not the correct color. Try again")
        lives -= 1
        screen.fill((255, 0, 0))  # Set screen color to red

    pygame.display.update()  # Update the screen to show the color
change

    pygame.time.wait(500)  # Keep color on screen for 0.5 second

    if lives == 0:
        game_over()
        pygame.quit()

#### Displays game over screen with the final score
def game_over():

    print("Game Over")
    print("Final Score:", score) # OBS! Work in progress
```

```
#### Position of squares on screen
def generate_x_y():
    x = (screen_width - (square_size * 4 + square_distance * 3)) // 2
# // = no decimals (integer division)
    y = (screen_height - square_size) // 2
    return x, y

#### Generate initial colors
draw_random_colors()

#### Game loop
running = True
while running:
   ##### Handle events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.MOUSEBUTTONUP:
   ##### Check if the player clicked on one of the color options
            mouse_pos = pygame.mouse.get_pos()
            x, y = generate_x_y()

            for i in range(4):
                option_rect = pygame.Rect(x, y, square_size,
square_size) # Making "hit-boxes" for the squares
                if option_rect.collidepoint(mouse_pos):
                    check_guess(i)
                    draw_random_colors()
                    break

                x += square_size + square_distance

   ##### Draw checkerboard background
    for row in range(screen_height // tile_size):
        for col in range(screen_width // tile_size):
            if (row + col) % 2 == 0:
                pygame.draw.rect(screen, (100, 100, 100), (col *
tile_size, row * tile_size, tile_size, tile_size))
            else:
                pygame.draw.rect(screen, (50, 50, 50), (col *
tile_size, row * tile_size, tile_size, tile_size))

   ##### Drawing squares for lives
    life_square_size = 20 # Note: width AND height
    life_square_distance = 10
    life_square_color = (255, 0, 0)
    life_square_x = 10 # Position x-axis
    life_square_y = 10 # Position y-axis
    for i in range(lives):
        life_square_rect = pygame.Rect(life_square_x, life_square_y,
life_square_size, life_square_size)
        pygame.draw.rect(screen, life_square_color, life_square_rect)
        life_square_x += life_square_size + life_square_distance

   ##### Drawing color options
    x, y = generate_x_y()

   ##### Making the squares appear in a line
    for i, color in enumerate(colors):
        width = color[3] # Extracting the width from color and making
it into its own variable
        option_rect = pygame.Rect(x + (square_size - width) // 2, y,
width, square_size) # Defines size of the squares
        pygame.draw.rect(screen, color[:3], option_rect) # Draws the
square with the specified colors
        x += square_size + square_distance

   ##### Update display
    pygame.display.update()

#### Quit game
pygame.quit()
```