# BLOCK-STACKING GAME

## Introduction

In recent years, the development of simple yet engaging 2D games has gained popularity due to their ease of implementation and entertainment value. This project, titled **"Color Stack Challenge"**, is a casual arcade-style game developed using **Python** and the **Pygame** library. It focuses on testing the player's timing and precision by requiring them to stack moving blocks as accurately as possible.

The idea behind this project is to demonstrate fundamental game development concepts such as graphics rendering, object-oriented programming, collision detection, user input handling, and score management. The game provides a colorful, interactive experience where players must align moving blocks with the existing stack. With each successful placement, the player's score increases, and the challenge intensifies as the stack grows taller.

This project serves both as a learning experience for understanding game mechanics and as a foundation for building more complex games in the future. It is an excellent example of how a basic concept can be turned into an engaging gameplay experience using Python and Pygame.

## Highlights of the Project

- **Interactive Gameplay**:
  A simple yet addictive block-stacking game that challenges players to time their moves precisely.
- **Colorful Visuals**:
  Blocks are rendered in vibrant random colors, making the game visually appealing and engaging.
- **Precision-Based Mechanics**:
  Players must align moving blocks with existing ones. A small margin of error is allowed, encouraging focus and coordination.
- **Real-Time Controls**:
  Instant response to player input (pressing the SPACE key to drop blocks), ensuring a smooth and dynamic gaming experience.
- **Scoring System**:
  Points are awarded for each successfully stacked block. The score is continuously displayed on the screen.

- **Game Over Detection**:
  If the player misaligns a block, the game ends, and the final score is shown — reinforcing the challenge.
- **Auto-Restart Feature**:
  After a game over, the game automatically restarts, keeping players engaged without needing to restart the application.
- **Object-Oriented Design**:
  Uses a Block class for better code structure, modularity, and future scalability.
- **Learning-Oriented Development**:
  Designed to help beginners understand key Pygame concepts like the game loop, rendering, event handling, and basic physics.

## What You Will Learn

- How to make a simple game with Python
- How to move and drop blocks
- How to take keyboard input (SPACE key)
- How to show score on the screen
- How to end the game when rules are broken
- How to restart the game automatically
- How to use colors and animations
- How to write clean code using classes

## Function Notes – Color Stack Challenge

### game_loop()

- The **main game function**
- Runs everything: block movement, input handling, drawing, scoring, and game over logic
- Loops until the user quits the game

### Block.__init__(self, x, y, color)

- **Constructor** of the Block class

- Sets the position (x, y), color, and movement direction of the block

**CODE:**
```
class Block:
    def __init__(self, x, y, color):
        self.x = x
        self.y = y
        self.color = color
        self.moving_right = True
```

**Block.move(self)**

- Moves the block left and right
- Changes direction when it hits the screen edge

**CODE:**
```
def move(self):
        if self.moving_right:
            self.x += BLOCK_SPEED
            if self.x + BLOCK_WIDTH >= WIDTH:
                self.moving_right = False
        else:
            self.x -= BLOCK_SPEED
            if self.x <= 0:
                self.moving_right = True
```

**Block.draw(self)**

- Draws the block on the screen using its position, size, and color

**CODE:**
```
def draw(self):
    pygame.draw.rect(screen, self.color, (self.x, self.y, BLOCK_WIDTH, BLOCK_HEIGHT))
```

**pygame.init()**

- Initializes all required Pygame modules before using them

**pygame.display.set_mode((WIDTH, HEIGHT))**

- Sets up the window (screen) with the given width and height

**pygame.display.set_caption("Color Stack Challenge")**

- Sets the title of the game window

**pygame.event.get()**

- Checks for all events (like key presses or window close)

**pygame.KEYDOWN + pygame.K_SPACE**

- Checks if the SPACE key is pressed to drop the block

**screen.fill(WHITE)**

- Fills the screen with a white background every frame

**pygame.draw.rect(screen, color, (x, y, width, height))**

- Draws a rectangle (used to draw each block)

**font.render(text, True, color)**

- Renders text (like score or "Game Over") to display on screen

**screen.blit(text, position)**

- Places the rendered text on the screen at the given position

**pygame.display.update()**

- Updates the display with everything drawn in the current frame

**pygame.time.delay(20)**

- Pauses for a short time (20 milliseconds) to control the game speed

**pygame.quit()**

- Exits the game and closes the window properly

**CODE:**

```
def game_loop():
    stack = []
    current_block = Block(WIDTH // 2 - BLOCK_WIDTH // 2, HEIGHT - 50,
random.choice(COLORS))
    score = 0
    running = True
```

```
 while running:
        screen.fill(WHITE)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
                if len(stack) == 0 or abs(stack[-1].x - current_block.x) <= 20:
                    stack.append(current_block)  # Add block to stack if aligned
                    score += 10
                    current_block = Block(WIDTH // 2 - BLOCK_WIDTH // 2, HEIGHT -
(len(stack) + 1) * 30, random.choice(COLORS))
                else:
                    text = font.render("Game Over! Score: " + str(score), True, BLACK)
                    screen.blit(text, (WIDTH // 2 - 150, HEIGHT // 2))
                    pygame.display.update()
                    pygame.time.delay(2000)
                    return game_loop()  # Restart game
for block in stack:
            block.draw()
        current_block.move()
        current_block.draw()
        score_text = font.render(f"Score: {score}", True, BLACK)
        screen.blit(score_text, (10, 10))
        pygame.display.update()
        pygame.time.delay(20)
pygame.quit()
game_loop()
```

# EXECUTION OF CODE

pip install pygame

```
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: pygame in c:\users\dell\appdata\roaming\pyth
on\python311\site-packages (2.6.1)
Note: you may need to restart the kernel to use updated packages.
```

import pygame

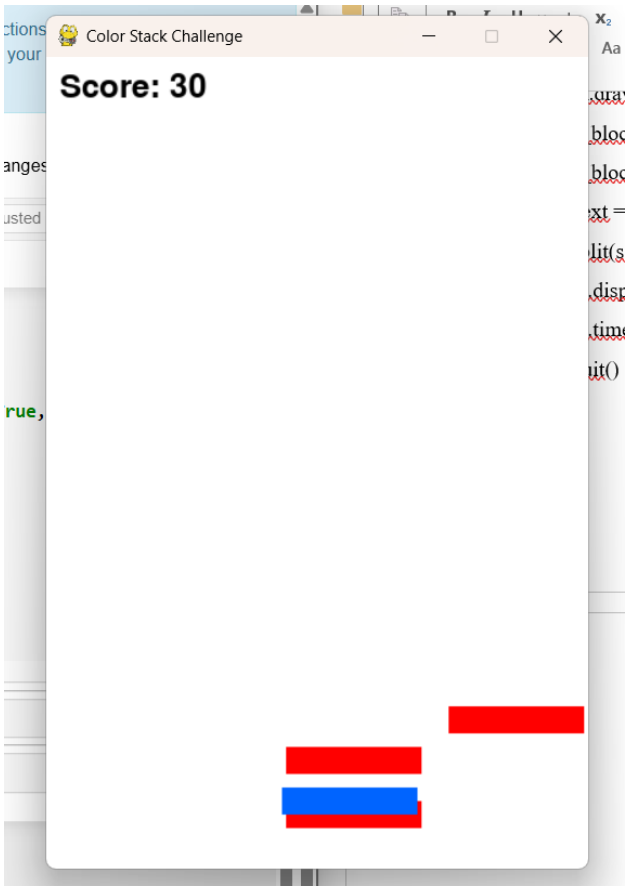
print("Pygame version:", pygame.__version__)

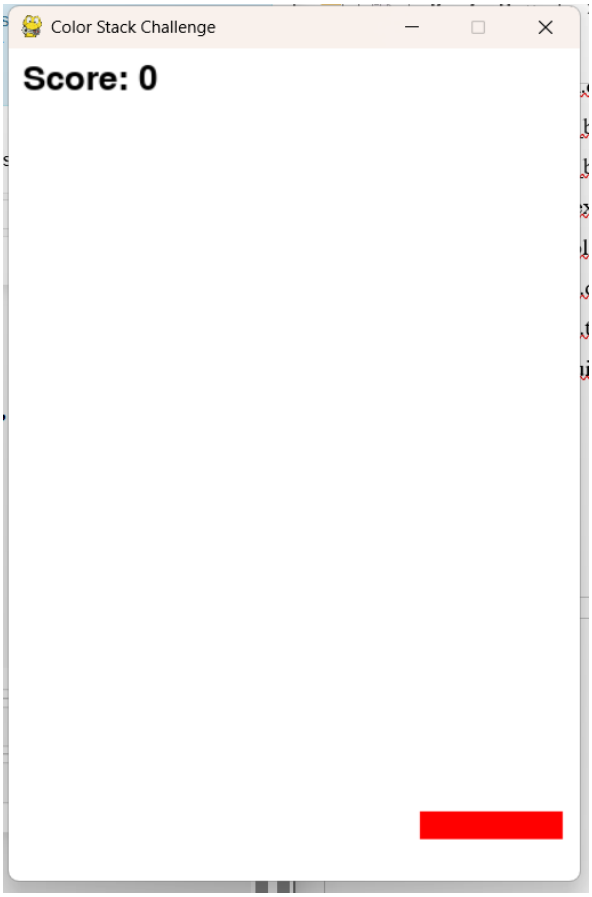
```
Pygame version: 2.6.1
```

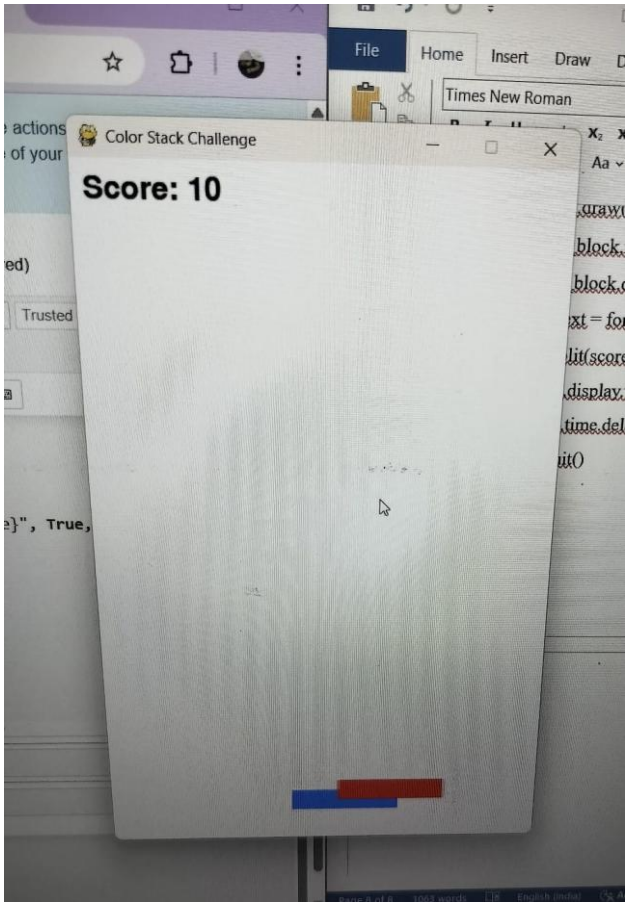
```
import pygame
import random
pygame.init()
WIDTH, HEIGHT = 400, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Color Stack Challenge")
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
COLORS = [(255, 0, 0), (0, 255, 0), (0, 100, 255), (255, 255, 0)]  # Red, Green, Blue, Yellow
BLOCK_WIDTH = 100
BLOCK_HEIGHT = 20
BLOCK_SPEED = 3
font = pygame.font.Font(None, 36)
class Block:
    def __init__(self, x, y, color):
        self.x = x
        self.y = y
```

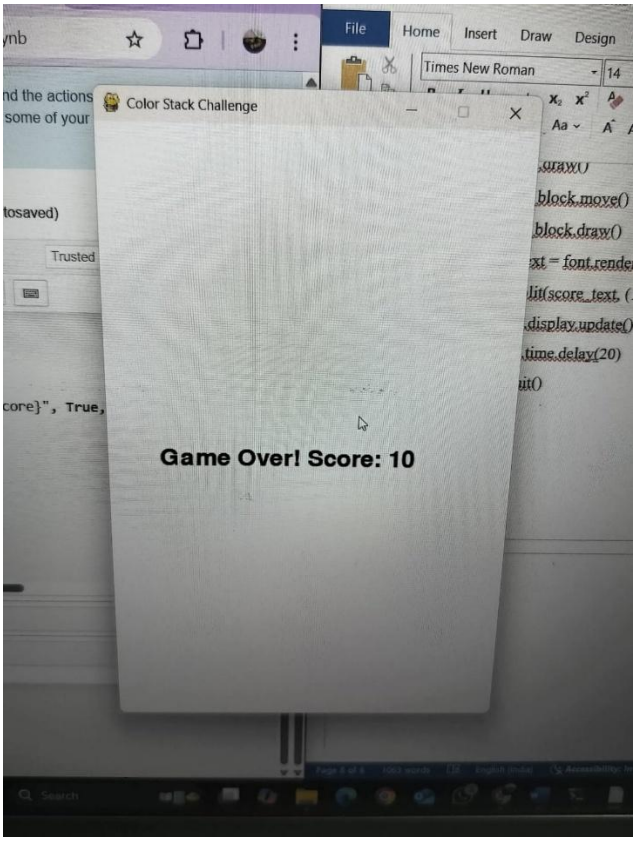
```
        self.color = color
        self.moving_right = True
    def move(self):
        if self.moving_right:
            self.x += BLOCK_SPEED
            if self.x + BLOCK_WIDTH >= WIDTH:
                self.moving_right = False
        else:
            self.x -= BLOCK_SPEED
            if self.x <= 0:
                self.moving_right = True
    def draw(self):
        pygame.draw.rect(screen, self.color, (self.x, self.y, BLOCK_WIDTH,
BLOCK_HEIGHT))
def game_loop():
    stack = []
    current_block = Block(WIDTH // 2 - BLOCK_WIDTH // 2, HEIGHT - 50,
random.choice(COLORS))
    score = 0
    running = True
    while running:
        screen.fill(WHITE)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
                if len(stack) == 0 or abs(stack[-1].x - current_block.x) <= 20:
                    stack.append(current_block)  # Add block to stack if aligned
                    score += 10
                    current_block = Block(WIDTH // 2 - BLOCK_WIDTH // 2, HEIGHT -
(len(stack) + 1) * 30, random.choice(COLORS))
                else:
                    text = font.render("Game Over! Score: " + str(score), True, BLACK)
```

```
            screen.blit(text, (WIDTH // 2 - 150, HEIGHT // 2))
            pygame.display.update()
            pygame.time.delay(2000)
            return game_loop()  # Restart game
        for block in stack:
            block.draw()
        current_block.move()
        current_block.draw()
        score_text = font.render(f"Score: {score}", True, BLACK)
        screen.blit(score_text, (10, 10))
        pygame.display.update()
        pygame.time.delay(20)
    pygame.quit()
game_loop()
```




Color Stack Challenge
Score: 0


Color Stack Challenge
Score: 10
File
Home
Insert
Draw
Times New Roman




# Conclusion:

The **Color Stack Challenge** project demonstrates how a simple game can be created using Python and the Pygame library. Through this project, we explored the basics of game development, including object-oriented programming, real-time user input handling, animation, and game logic.

By building this game, we gained hands-on experience with key concepts like the game loop, event handling, collision checking, and rendering graphics on the screen. The project also helped improve logical thinking, problem-solving, and code structuring skills.

Overall, this project serves as a great foundation for beginners in game development and opens up possibilities for more advanced features and games in the future.