

Task 1-12.

Task 12: Stimulate Gaming concepts using pygame. Date:- 13/10/25

Aim:- To simulate gaming concepts using Pygame.

Problem 1: Write a Python program to create a snake game using Pygame package.

Algorithm:-

1. Import Pygame package and initialize it.
2. Define the window size and title.
3. Create a snake class which initializes the fruit position color and movement.
4. Create a fruit class which initializes the fruit position and color.
5. Create a function to check if the snake collides with the fruit and increase the score.
6. Create a function to check if the snake collides with the window and end the game.
7. Create a function to update the snake position based on the user input.
8. Create a function to update the game display and draw the snake and fruit.
9. Create a game loop to continuously update the game display, snake position, and check for collisions.
10. End the game if the user quits or the snake collides with the window.

Program.

importing libraries.

import pygame.

import time.

import random.

snake_speed = 15.

window size.

window_x = 720.

window_y = 480.

defining colours.

black = pygame.color(0,0,0)

white = pygame.color(255,255,255)

red = pygame.color(255,0,0)

green = pygame.color(0,255,0)

blue = pygame.color(0,0,255)

initializing pygame.

pygame.init()

initialize game window.

pygame.display.set_caption('Grecks for Grecks Snakes').

2-10-15-2000

Output:-

Score 100

```

game - window =
pygame - display.set - caption ('Blocks for Brooks Snake')
game - window =
pygame - display.set - mode ((window-x, window-y))
# FPS (frames per second) controller.
FPS = pygame.time.Clock()
# defining snake default position.
Snake - position = [100, 50].
# defining first 4 blocks of snake body.
Snake - body = [[100, 50],
                 [90, 50],
                 [80, 50],
                 [70, 50]
                ]
# fruit position.
fruit - position
fruit - position = [random.randrange(1, (window-x//100)*10),
                    random.randrange(1, (window-y//100)*10)]
# Game over condition.
if Snake - position [0] < 0 or
Snake - position [0] > window - x - 10:
    game - over ()
if Snake - position [1] < 0 or
Snake - position [1] > window - y - 10:
    game - over ()
# touching the snake body.
for block in Snake - body [1:]:
    if Snake - position [0] == block [0] and
Snake - position [1] == block [1]:
        game - over ()
# displaying score continuously.
show - score (1, white, 'Time New Roman', 20)
# Refresh game screen.
pygame - display.update ()
# frame per second / refresh rate
FPS - tick (Snake - speed)

```


Problem 2 : Write a Python program to develop a chess board using Pygame.

Algorithm:-

1. Import Pygame and initialize it.
2. Set screen size and title.
3. Define colors for the board and pieces.
Define a function to draw the board by looping over rows and columns and drawing squares of different colors.
4. Define a function to draw the pieces on the board by loading images for each piece and placing them on the corresponding square.
5. Define the initial state of the board as a list of lists containing the pieces.
6. Draw the board and pieces on the screen.
7. Start the game loop.

Program:-

```
import pygame.  
# initialize pygame,  
pygame.init()  
# set screen size and title.  
screen_size = (640, 640)  
screen =  
pygame.display.set_mode(screen_size)  
pygame.display.set_caption('chess board')  
# Define colors.  
black = (0, 0, 0)  
white = (255, 255, 255)  
brown = (153, 76, 0)  
# Define function to draw the board.  
def draw_board():  
    for row in range(8):  
        for col in range(8):  
            square_color = white if (row+col)%2 == 0 else brown  
            square_rect = pygame.Rect(col*80, row*80, 80, 80)  
            pygame.draw.rect(screen, square_color, square_rect)  
# Define function to draw the pieces.  
def draw_pieces(board):  
    piece_images = {
```

Programs : write a linked program to find a
 that board is full or not.

Algorithm :

1. To test whether the board is full or not.
2. If board is full then return true.
3. If board is not full then return false.

Write a function to check the board is full or not.

Code :

1. To test whether the board is full or not.

2. If board is full then return true.

3. If board is not full then return false.

4. To test whether the board is full or not.

5. If board is full then return true.

6. If board is not full then return false.

7. To test whether the board is full or not.

8. If board is full then return true.

9. If board is not full then return false.

10. To test whether the board is full or not.

```

'r':
pygame.image.load('images/rook.png')
'n':
pygame.image.load('images/knight.png')
'b':
pygame.image.load('images/bishop.png')
'q':
pygame.image.load('images/queen.png')
'k':
pygame.image.load('images/pawn.png')
}
for row in range(8):
    for col in range(8):
        piece = board[row][col]
        if piece != ' ':
            piece_image = piece_images[piece]
            piece_rect = pygame.Rect(col*80, row*80, 80, 80)
            screen.blit(piece_image, piece_rect)

# start game loop.
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            exit()
    pygame.display.update()

```

Result:- Thus the program for ~~pygame~~ is executed and verified successfully.

VEL TECH	
EM No	
PERFORMANCE (5)	12
ANALYSIS (5)	5
AV VOICE (5)	4
ORD (5)	14
TOTAL (20)	
DATE	