HRT TECH SOLUTIONS

Name – Aman Kumar

Email – aman9155635728@gmail.com

TASK 1 : SNAKE GAME

Code :

import pygame

import random

pygame.init()

# Constants

SCREEN\_WIDTH, SCREEN\_HEIGHT = 800, 800

CELL\_SIZE = 20

WHITE = (255, 255, 255)

GREEN = (0, 255, 0)

RED = (255, 0, 0)

# Initialize variables

screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

pygame.display.set\_caption('Snake Game')

clock = pygame.time.Clock()

snake\_pos = [(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2)]

snake\_direction = 'RIGHT'

food\_pos = (random.randrange(1, SCREEN\_WIDTH // CELL\_SIZE) \* CELL\_SIZE,random.randrange(1, SCREEN\_HEIGHT // CELL\_SIZE) \* CELL\_SIZE)

# Game loop

while True:

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            pygame.quit()

            quit()

        keys = pygame.key.get\_pressed()

        for key in keys:

            if keys[pygame.K\_LEFT] and snake\_direction != 'RIGHT':

                snake\_direction = 'LEFT'

            elif keys[pygame.K\_RIGHT] and snake\_direction != 'LEFT':

                snake\_direction = 'RIGHT'

            elif keys[pygame.K\_UP] and snake\_direction != 'DOWN':

                snake\_direction = 'UP'

            elif keys[pygame.K\_DOWN] and snake\_direction != 'UP':

                snake\_direction = 'DOWN'

    # Move the snake

    head\_x, head\_y = snake\_pos[0]

    if snake\_direction == 'RIGHT':

        head\_x += CELL\_SIZE

    elif snake\_direction == 'LEFT':

        head\_x -= CELL\_SIZE

    elif snake\_direction == 'UP':

        head\_y -= CELL\_SIZE

    elif snake\_direction == 'DOWN':

        head\_y += CELL\_SIZE

    snake\_pos.insert(0, (head\_x, head\_y))

    # Check for collisions

    if head\_x >= SCREEN\_WIDTH or head\_x < 0 or head\_y >= SCREEN\_HEIGHT or head\_y < 0:

        pygame.quit()

        quit()

    if snake\_pos[0] in snake\_pos[1:]:

        pygame.quit()

        quit()

    # Check if the snake eats the food

    if snake\_pos[0] == food\_pos:

        food\_pos = (random.randrange(1, SCREEN\_WIDTH // CELL\_SIZE) \* CELL\_SIZE,random.randrange(1, SCREEN\_HEIGHT // CELL\_SIZE) \* CELL\_SIZE)

    else:

        snake\_pos.pop()

    # Draw everything

    screen.fill(WHITE)

    for pos in snake\_pos:

        pygame.draw.rect(screen, GREEN, (pos[0], pos[1], CELL\_SIZE, CELL\_SIZE))

    pygame.draw.rect(screen, RED, (food\_pos[0], food\_pos[1], CELL\_SIZE, CELL\_SIZE))

    pygame.display.update()

    clock.tick(10)

1. **Importing Libraries:**
   * **pygame** is imported for building the game.
2. **Initializing Pygame:**
   * **pygame.init()** initializes the Pygame modules.
3. **Constants:**
   * **SCREEN\_WIDTH** and **SCREEN\_HEIGHT** define the dimensions of the game window.
   * **CELL\_SIZE** determines the size of each cell in the game grid.
   * Colors like **WHITE**, **GREEN**, and **RED** are defined in RGB format.
4. **Initializing Variables:**
   * **screen** is created with the specified dimensions.
   * The snake's initial position is set in the middle of the screen.
   * **snake\_direction** keeps track of the snake's current direction.
   * **food\_pos** holds the position of the food item.
5. **Game Loop:**
   * The game runs in an infinite loop (**while True:**) to keep the game running until the player quits.
   * The loop processes events like quitting the game or changing the snake's direction based on user input.
6. **Handling User Input:**
   * The code checks for user input using **pygame.key.get\_pressed()** to change the direction of the snake.
   * Depending on the input, the **snake\_direction** variable is updated to control the movement of the snake.
7. **Moving the Snake:**
   * The snake's position is updated based on its current direction.
   * The new head position is calculated, and the new head is added to the snake's position.
   * If the snake eats the food, it grows longer. If not, the last segment of the snake is removed to simulate movement.
8. **Collision Detection:**
   * The code checks for collisions with the game window boundaries and with the snake itself.
   * If the snake collides with the window boundaries or itself, the game exits.
9. **Food Generation:**
   * If the snake's head position matches the food position, a new random position for the food is generated.
   * If the snake eats the food, it grows longer; otherwise, it stays the same length.
10. **Drawing the Game:**
    * The game screen is filled with a white background.
    * The snake and food are drawn as rectangles on the screen using the Pygame **pygame.draw.rect()** function.
    * The screen is updated with the new positions of the snake and food, creating the illusion of movement.
11. **Frame Rate Control:**
    * **clock.tick(10)** limits the frame rate to 10 frames per second, regulating the speed of the game.

TASK 2 :

**Identify a Problem Statement Within your Chosen Domain and devise a solution for it. Then, articulate the problem and your proposed solution.**

Problem: In the domain of Python programming, one common challenge faced by developers is the inefficient management and deployment of dependencies, especially when working on multiple projects simultaneously. Managing different package versions, ensuring compatibility, and avoiding conflicts can lead to time-consuming troubleshooting and hinder the development process.

Proposed Solution:

Solution: The development of a virtual environment management tool specifically tailored for Python projects. This tool will allow developers to create isolated environments for each project, ensuring that dependencies are neatly organized, version conflicts are minimized, and compatibility issues are reduced. Additionally, the tool will provide intuitive commands for environment creation, activation, and deactivation, simplifying the process of managing multiple projects with different dependencies.

**Articulation of the Problem:**

Python developers frequently work on various projects, each requiring specific packages and libraries with different versions. When these dependencies clash, it can lead to errors, bugs, and project delays. Managing these dependencies within a single global environment can be chaotic, as updating or installing packages for one project might inadvertently affect another. This situation is particularly challenging when developers need to collaborate on projects or when they switch between projects frequently.

The lack of an efficient dependency management system can lead to frustration, decreased productivity, and, in some cases, discourage developers from exploring new libraries or tools due to concerns about breaking existing projects.

**Articulation of the Solution:**

The proposed solution involves the creation of a user-friendly virtual environment management tool specifically designed for Python projects. This tool will enable developers to create isolated environments for each project, ensuring that dependencies are kept separate and do not interfere with one another. Each virtual environment will have its own Python interpreter and package installations, allowing developers to work on different projects with distinct requirements without worrying about version conflicts or compatibility issues.

The tool will provide simple and intuitive commands for creating new virtual environments, activating and deactivating them, and installing or updating packages within a specific environment. Developers can easily switch between projects by activating the corresponding virtual environment, streamlining the process of managing multiple projects with varying dependencies.

By implementing this virtual environment management tool, Python developers will experience improved workflow efficiency, reduced debugging time, and increased confidence in exploring and experimenting with new packages and libraries. This solution will foster a more organized and productive development environment, ultimately enhancing the overall Python programming experience for developers.