Name: ANJALA FARHA
*Email id: **anjalafarhak@gmail.com***

SNAKE GAME

ABSTRACT

The Nokia inspired Snake game was implemented on 8051 platform. The game is built on the AT89551 Microcontroller. The game uses an 8 8 LED dot matrix display and five way keys for user interface. A dedicated delay settings key provides one touch access to the delay settings mode where the speed of the Snake can be adjusted.

The Snake game project in Python is a simple game that involves moving a snake around a screen to eat food while avoiding collisions with walls or the snake's own body. The player controls the snake using arrow keys, and the snake's length increases by one unit each time it eats food.

The game is implemented using Python's Pygame library, which provides a set of functions for creating game graphics and handling user input. The game consists of a game loop that updates the position of the snake and the food, checks for collisions, and renders the game graphics.

To implement the game, we start by creating the game window and loading the game graphics. We then define the Snake and Food classes, which contain the logic for moving the snake and the food. The game loop updates the positions of the snake and the food, and checks for collisions using collision detection algorithms.

The game also includes a scoring system that keeps track of the player's score, which is incremented each time the snake eats food. The game ends when the snake collides with a wall or its own body, at which point the player's score is displayed on the screen.

Overall, the Snake game project in Python is a fun and engaging project that provides an introduction to game programming and Python programming concepts such as classes, objects, and functions.

Concepts for Snake game project:

1. Basic Python programming concepts such as loops, conditionals, functions, and classes.

2. Pygame library for game development in Python.

3. Game development concepts such as game loop, collision detection, and game graphics.

4. Object-oriented programming (OOP) concepts such as classes, objects, and inheritance.

5. Working with user input, such as keyboard input for controlling the snake.

6. Algorithmic thinking and problem-solving skills for designing game logic and mechanics.

7. Debugging and troubleshooting skills for identifying and fixing errors in the code.

Optional advanced topics such as AI agents for controlling the snake or implementing networked multiplayer mode.

I. PyGame - Py game is a cross-platform set of python modules designed for writing video games.

It includes computer graphics and sound libraries designed to be used with the Python Programming language.

- To install the library, you can use pip installer from the command line:

```
pip install pygame

import pygame
```

II.Python time module

Python has a module named time to handle time-related tasks.

To use functions defined in the module, we need to import the module first. Here's how:

```
import time
```

III.Python random module

Python has a built-in module that you can use to make random numbers,

```
import random
```

PROBLEM STATEMENT

Create a classic snake game using Python where the player controls a snake that moves around a rectangular board. The game should have the following features:

1.The snake should start with a length of 1 and increase in length every time it eats a food item.

2.The snake should die if it hits a wall or runs into its own tail.

3.The game should have a score counter that increases every time the snake eats a food item.

4.The game should end when the snake dies and the player should have the option to restart the game.

5.The game should have a user-friendly interface and easy-to-understand controls.

The game should be visually appealing and feature sound effects.

6.The game should be coded in Python using appropriate data structures and algorithms.

Overall, the goal of this project is to create a fun and engaging game that provides a good coding challenge while also being entertaining for players.

The main classes that can be used are:

a. Snake

b. Cell

c. Board

d. Game

PROPOSED SOLUTION

1. Import necessary modules: First, import the required modules, such as the 'pygame' module for creating the game window and handling game events, and 'random module' for generating random positions of food for the snake to eat.

```
import pygame

import random
```

2. Initialize the game window: Set the size of the game window and create a window using the 'pygame' module.

```
# Set the size of the game window

window_width = 500

window_height = 500


# Create the game window

pygame.init()

game_window = pygame.display.set_mode((window_width, window_height))

pygame.display.set_caption("Snake Game")
```

3. Define the Snake class: Define a 'Snake' class to handle the movement and position of the snake.

```
class Snake:

    def __init__(self):

        self.body = [(window_width//2, window_height//2)]

        self.direction = "right"

        self.size = 10

        self.score = 0

    def move(self):

        x, y = self.body[0]
        if self.direction == "right":
            x += self.size
        elseif self.direction == "left":
            x -= self.size
        elseif self.direction == "up":
            y -= self.size
        elseif self.direction == "down":
            y += self.size
        self.body.insert(0, (x, y))
        self.body.pop()
```

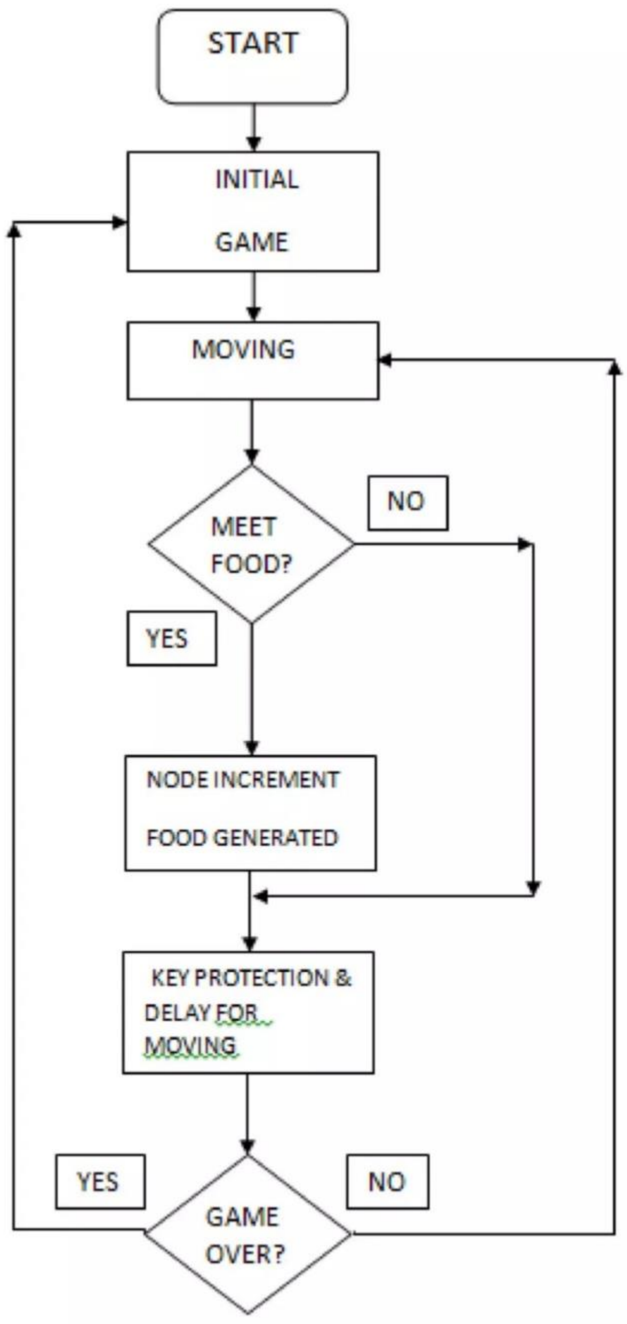
4. Define the Food class: Define a 'Food' class to handle the position of the food that the snake eats.

```
class Food:
    def __init__(self):
        self.position = (random.randint(0, window_width//10)*10, random.randint(0, 
window_height//10)*10)
        self.size = 10

    def respawn(self):
        self.position = (random.randint(0, window_width//10)*10, random.randint(0, 
window_height//10)*10)
```

5. Define the game loop: Create a game loop that handles game events and updates the game state.

```
snake = Snake()
food = Food()
running = True
clock = pygame.time.Clock()
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elseif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_RIGHT and snake.direction != "left":
                snake.direction = "right"
            elseif event.key == pygame.K_LEFT and snake.direction != "right":
                snake.direction = "left"
            elseif event.key == pygame.K_UP and snake.direction != "down":
                snake.direction = "up"
            elseif event.key == pygame.K_DOWN and snake.direction != "up":
                snake.direction = "down"
    snake.move()
    if snake.body[0] == food.position:
        snake.score += 1
        snake.body.append(snake.body[-1])
        food.respawn()
    game_window.fill((0, 0, 0))
    pygame.draw.rect(game_window, (255, 0, 0), (food.position[0], food.position[1], food.size, food.size))
    for i, (x, y) in enumerate(snake.body):
        color = (255, 255, 255) if i == 0 else (0, 255, 0)
        pygame.draw.rect(game_window, color, (x, y, snake.size, snake.size))
    pygame.display.update()
    clock.tick(10)
pygame.quit()
```

SYSTEM ARCHITECTURE



Game initialization: The game starts with initializing the game window and setting its size, and creating the Snake and Food objects.

Game loop: The game loop handles game events, such as key presses or quitting the game, and updates the game state. The loop runs until the player loses the game or quits the game.

Snake movement: The movement of the snake is updated in each iteration of the game loop. The direction of the snake is controlled by the player's key presses.

Collision detection: The game checks for collisions between the snake and the walls of the game window or the food. If the snake collides with the walls, the game is over. If the snake collides with the food, the score is increased, the length of the snake is increased, and the food is respawned at a new random location.

Game over: If the snake collides with the walls of the game window, the game is over, and the player is presented with a message indicating the game is over, and the final score is displayed.

User interface: The game window displays the game state, including the score, the position of the snake and food, and the game over message.

Score keeping: The game keeps track of the player's score and displays it on the game window.

SOURCE CODE OF THE ALGORITHM

```
Import pygame

Import time

Import random

Pygame.init()

White = (255, 255, 255)

Black = (0, 0, 0)

Red = (213, 50, 80)

Green = (0, 255, 0)

Blue = (50, 153, 213)

Width = 600

Height = 400


Dis = pygame.display.set_mode((width, height))

Pygame.display.set_caption(‘Snake Game’)
```

```
Clock = pygame.time.Clock()

Snake_block = 10

Snake_speed = 15

Font_style = pygame.font.SysFont(None, 30)

Def message(msg, color):

   Mesg = font_style.render(msg, True, color)

   Dis.blit(mesg, [width / 6, height / 3])


Def gameLoop():

   Game_over = False

   Game_close = False

   X1 = width / 2

   Y1 = height / 2

   X1_change = 0

   Y1_change = 0


   Foodx = round(random.randrange(0, width – snake_block) / 10.0) * 10.0

   Foody = round(random.randrange(0, height – snake_block) / 10.0) * 10.0

   While not game_over

      While game_close == True:

         Dis.fill(blue)

         Message("You Lost! Press Q-Quit or C-Play Again", red)

         Pygame.display.update()


         For event in pygame.event.get():

            If event.type == pygame.KEYDOWN:

               If event.key == pygame.K_q:

                  Game_over = True

                  Game_close = False
            If event.key == pygame.K_c:
                gameLoop()

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        game_over = True
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_LEFT:
            x1_change = -snake_block
            y1_change = 0
        elif event.key == pygame.K_RIGHT:
            x1_change = snake_block
            y1_change = 0
        elif event.key == pygame.K_UP:
            y1_change = -snake_block
            x1_change = 0
        elif event.key == pygame.K_DOWN:
            y1_change = snake_block
            x1_change = 0

if x1 >= width or x1 < 0 or y1 >= height or y1 < 0:
    game_close = True

x1 += x1_change
y1 += y1_change
dis.fill(blue)
pygame.draw.rect(dis, green, [foodx, foody, snake_block, snake_block])
pygame.draw.rect(dis, black, [x1, y1, snake_block, snake_block])
pygame.display.update()
```

```
        if x1 == foodx and y1 == foody:
            print(“Yummy!!”)
        clock.tick(snake_speed)
    pygame.quit()
    quit()
gameLoop()
```

CODE REPO LINK

EXAMPLE

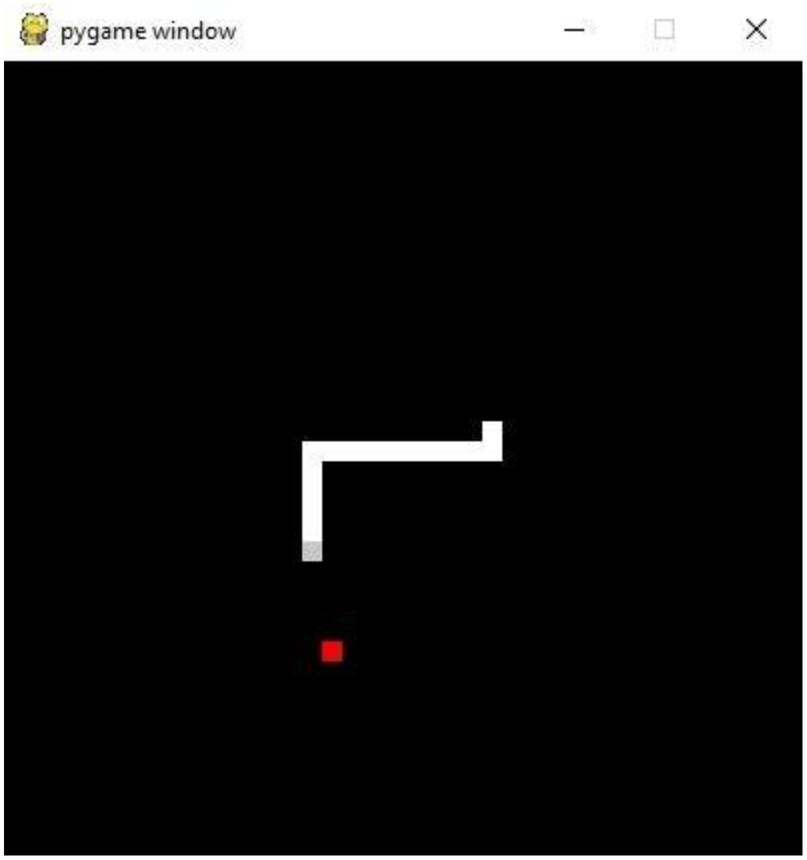
```
Import pygame
Import time
Import random

# initialize pygame
Pygame.init()

# set up the game window
Window_width = 640
Window_height = 480
Game_window = pygame.display.set_mode((window_width, window_height))
Pygame.display.set_caption(‘Snake Game’)

# set up the colors
```

```
White = (255, 255, 255)

Black = (0, 0, 0)

Red = (255, 0, 0)

Green = (0, 255, 0)

Blue = (0, 0, 255)


# set up the game clock

Clock = pygame.time.Clock()

# set up the font

Font = pygame.font.SysFont(None, 30)

# set up the snake

Snake_size = 10

Snake_speed = 15

Snake_list = []

Snake_length = 1

Snake_x = window_width / 2

Snake_y = window_height / 2

Snake_x_change = 0

Snake_y_change = 0


# set up the food

Food_size = 10

Food_x = round(random.randrange(0, window_width – food_size) / 10.0) * 10.0

Food_y = round(random.randrange(0, window_height – food_size) / 10.0) * 10.0
```

```
# set up the game loop
Game_over = False
While not game_over:
   # handle events
   For event in pygame.event.get():
      If event.type == pygame.QUIT:
         Game_over = True
      If event.type == pygame.KEYDOWN:
         If event.key == pygame.K_LEFT:
            Snake_x_change = -snake_size
            Snake_y_change = 0
         Elif event.key == pygame.K_RIGHT:
            Snake_x_change = snake_size
            Snake_y_change = 0
         Elif event.key == pygame.K_UP:
            Snake_y_change = -snake_size
            Snake_x_change = 0
         Elif event.key == pygame.K_DOWN:
            Snake_y_change = snake_size
            Snake_x_change = 0
   # move the snake
   Snake_x += snake_x_change
   Snake_y += snake_y_change
   # check if the snake has hit the wall
```

```
    If snake_x < 0 or snake_x > window_width – snake_size or snake_y < 0 or
snake_y > window_height – snake_size:
        Game_over = True
    # check if the snake has hit itself
    Head = []
    Head.append(snake_x)
    Head.append(snake_y)
    Snake_list.append(head)
    If len(snake_list) > snake_length:
        Del snake_list[0]
    For segment in snake_list[:-1]:
        If segment == head:
            Game_over = True
    # check if the snake has eaten the food
    If snake_x == food_x and snake_y == food_y:
        Food_x = round(random.randrange(0, window_width – food_size) / 10.0) *
10.0
        Food_y = round(random.randrange(0, window_height – food_size) / 10.0) *
10.0
        Snake_length += 1
    # draw the game window
    Game_window.fill(black)
    Pygame.draw.rect(game_window, green, [food_x, food_y, food_size,
food_size])
    For segment in snake_list:
        Pygame.draw.rect(game_window, white, [segment[0], segment[1], snake_size, snake_size])
    Pygame.display.update()
    # set up the game clock
    Clock.tick(snake_speed)
# end
```

(Snake game window with a black background and a white snake moving around the screen. there will also be a red food block that the snake can eat to increase its length. the snake movement is controlled by the arrow keys, and if the snake hits a wall or itself, the game will end and a "game over" message will be displayed.)



ANALYSIS AND FINDINGS

Object-Oriented Programming: Implementing the Snake Game using object-oriented programming can make the code modular and easy to maintain. The game can be broken down into various classes such as the Snake, Food, Game, etc., each with their own specific functionality.

Pygame Library: Pygame library in Python can be used to create the game window, handle user input, and render graphics on the screen. Pygame also provides built-in functions for collision detection, which can be useful in detecting when the snake collides with the food or the walls.

Game Loop: The game loop is an essential component of any game project. In the Snake Game project, the game loop is responsible for updating the position of the snake and the food, checking for collisions, and rendering the game on the screen.

Difficulty Levels: Implementing different difficulty levels, such as easy, medium, and hard, can make the game more challenging and engaging for players. This can be achieved by adjusting the speed of the snake or the number of obstacles on the screen.

High Scores: Adding a feature to track and display high scores can motivate players to keep playing the game and beat their previous scores. This can be implemented using file I/O to store and retrieve high scores.

User Interface: The user interface is an essential component of any game project. In the Snake Game project, designing a simple and intuitive user interface can enhance the user experience.

The Snake Game project using Python can be an excellent way to learn and practice various programming concepts, such as object-oriented programming, game development, and user interface design.

CONCLUSION

The Snake Game project using Python is a classic arcade-style game that challenges players to maneuver a snake around the game board to collect food without colliding with obstacles. The Snake Game project using Python is a great way to learn and practice various programming concepts such as object-oriented programming, game development, and user interface design. By breaking down the game into classes and functions, developers can create modular and maintainable code. The Pygame library can be used to handle game graphics and user input, making it easier to create an engaging game experience. The project uses object-oriented programming concepts to create modular and maintainable code, and the Pygame library is used to handle game graphics and user input. The game loop is an essential component of the project, responsible for updating the snake’s position and rendering graphics on the screen. Difficulty levels and high scores are implemented to make the game more challenging and engaging. Although the Snake Game project is a fun and exciting game, there are areas where it could be improved. For instance, adding sound effects and background music can enhance the game’s overall experience. Additionally, implementing different game modes or power-ups can add variety and keep players engaged for longer. Overall, the Snake Game project using Python is an excellent opportunity for developers to learn and practice fundamental programming concepts while creating a fun and engaging game.

REFERENCE

1-"How to Build a Snake Game in Python Using Pygame" by Real Python - this blog provides a step-by-step guide on how to build a Snake Game project using Python and Pygame. It covers topics such as game design, user input, and collision detection.

2-"Creating a Snake Game in Python" by Code with Ania Kubów - this blog covers how to create a Snake Game project using Python, Pygame, and object-oriented programming concepts. It covers topics such as creating a game window, handling user input, and game logic.

3-"Snake Game in Python" by PythonForBeginners - this blog provides a simple and easy-to-follow tutorial on how to create a basic Snake Game project using Python and Pygame.

4-"Python Snake Game Tutorial – Build a Classic Snake Game in Python from Scratch" by Tech With Tim - this blog covers how to create a Snake Game project using Python, Pygame, and object-oriented programming concepts. It covers topics such as creating the snake object, handling collision detection, and game logic.