**COMPUTER VISION**

**CS – GY 6643**

**PROJECT: SNAKE GAME**

**TEAM:**

**HARSH JALUTHARIA – *hj2607***

**MUDIT NIGAM – *mn3439***

**RHUTHVIK DENDUKURI – *rd3377***

Introduction

# Background

The original Snake game, popularized by Nokia's monochrome mobile phones in the late 1990s, is a classic arcade-style video game. The objective is to control a continuously growing snake by guiding it to eat food items while avoiding collisions with its own body or the boundaries. As the snake grows longer with each piece of food consumed, the challenge lies in navigating through an increasingly crowded and confined space.

The game's simplicity and portability contributed to its enduring popularity, making it an ideal testbed for computer vision projects focused on tracking movement, detecting collisions, and developing advanced features like obstacle avoidance or path planning algorithms.

# Motivation

A great chance to create and test computer vision algorithms in a controlled setting is the Snake game. It presents various non-trivial issues for computer vision systems despite having straightforward gameplay mechanisms. Robust vision algorithms are needed to identify food items, follow the snake's movement accurately, and detect impacts with objects or the snake's own body. Furthermore, the continuous nature of the game and its rising complexity as the snake lengthens can be utilized to assess how well vision techniques operate and how scalable they are under different circumstances. Starting with this well-defined and well-known game allows researchers to concentrate on creating unique vision techniques without being distracted by intricate rendering or physics models.

# Objectives

## Implement a Snake Game

The primary objective is to develop the core mechanics of the snake game, encompassing essential elements such as snake movement, food and hurdle generation and a scoring system. By recreating the classic gameplay, we aim to provide a familiar yet enjoyable experience for players.

## Integrate Computer Vision

Leveraging computer vision algorithms, we seek to detect objects within the game environment and enable players to control the snake’s movements using real-world objects captured by the system’s web camera. This integration adds a unique layer of interactivity to the game, allowing players to engage with the virtual world in innovative ways.

## Enhance Gameplay Experience

The goal is to create an engaging and intuitive gameplay experience by seamlessly blending computer vision technology with the classic game mechanics.

## Scalability and Performance

It is crucial to design the system to be scalable and efficient, capable of running in real-time in standard hardware while delivering a smooth experience. By optimizing algorithms and leveraging hardware acceleration where possible, we aim to ensure that the game performs reliably across a range of devices and environments.

Project Overview

# Description

This project reimagines the Snake Game by integrating computer vision. Players control the snake, collecting food and avoiding obstacles. What sets it apart is the ability to control the snake using real-world objects detected through a camera feed. Computer vision algorithms analyze the feed in real-time, translating object positions into commands for the snake's movement. This immersive gameplay experience blends reality with virtuality, offering dynamic interaction. Whether navigating household items or custom obstacles, players enjoy a fusion of nostalgia and innovation.

# Technologies Used

1. Python
2. OpenCV
3. Numpy
4. Tkinter
5. Random Module

# CV techniques Used

## Color – based Segmentation

This involves segmenting the camera feed based on color to isolate the objects of interest from the scene. This is typically done using color thresholding techniques, where pixels within a certain color range are considered part of the object, while the others are discarded. In our scenario, this process is implemented using the range from the color codes “(29, 86, 18)” (lowest green HSV code) to “(93, 255, 255)” (highest green HSV code).

## Preprocessing

Before further analysis, the segmented image may undergo preprocessing steps like erosion and dilation, which are further discussed, to enhance the object boundaries and remove noise. These operations helps refine the object masks and improve accuracy of subsequent processing steps.

## Contour detection

Once the preprocessed image is computed, contours are detected using “cv2.findContours()” function. Contours represent the boundaries of connected regions within the segmented image and are essential for identifying and analyzing objects.

The input parameters of this function take an image, contour retrieval mode and contour approximation method. For our necessity, the contour retrieval mode is set to cv2.RETR\_EXTERNAL which retrieves only the external contours (outer boundary of the object) and the contour approximation method is set to cv2\_CHAIN\_APPROX\_SIMPLE which compresses horizontal, vertical and diagonal segments leaving only their endpoints.

Another way to find the contours is y Using Hough Transform but it had several complexities.

Using Hough Transform with edge orientation was a challenge we faced in this project. This method involves detecting edges and computing edge orientations before applyting hough transform. This is a complex computational approach as the frames give a continuous input to the system to which hough transform has to be applied and the result has to be provided within no time. The accuracy of edge detection affects the performance of hough transform.

In this scenario, the factors like light conditions, object textures and noise levels play a strong role in quality of edge detection. In addition, edge detection is sensitive to noise in the image. Though a gaussian blur is used to smoothen the image and lessen the noise, the computed edges were not sufficiently clear for the hough transform to perform with accuracy. Also, performing edge detection and orientation estimation for every frame in such a real – time scenario imposes significant computational overhead.

In conclusion, since such edge detection and orientation need algorithms with strict time constraints, “cv2.findContours” performed both accurately and faster.

## Object Localization and Identification

Object localization and identification are critical components that enable precise interaction between the virtual snake and real-world items. Through meticulous contour analysis, the system extracts vital geometric information from observed objects, including their sizes, shapes, and spatial orientations. This analysis ensures the accurate localization and tracking of objects within the gaming environment, facilitating realistic interaction with the virtual snake.

Leveraging the derived contour information, the game dynamically responds to the player's actions, offering seamless and captivating gameplay experiences. By applying additional criteria to filter out extraneous contours and identify objects of interest, the system distinguishes between valid objects and noise or background elements. This process enhances gameplay dynamics by ensuring accurate interaction between the virtual snake and detected objects, delivering a more immersive and responsive gaming experience for players.

# Implementation Details

## Game Loop/ Logic

### Initialization:

Certain variables have been initialized to establish the initial state of the game. Namely:

* + - Snake Length: initial length of the snake at the start of the game.
    - Score: initial score of the game is set to 0
    - Number of Hurdles: Initializing the number of hurdles to appear per game on the game frame.
    - Win Score: Acts as a win criterion. The score to be achieved to win the game.
    - Food Image: Assigning an image to the food in the game, the image file path is stored.
    - Hurdle Image: Assigning an image to the hurdle in the game, the image file path is stored.

### Main Loop:

* + Continuous Frame Capture: The main loop continuously captures frames from the camera feed to provide real-time input for the game environment.
  + Color – Based Segmentation: Color – based segmentation is applied to each captured frame to isolate objects of interest, particularly snake’s head. This involves pixels within a specified color range that likely belongs to the snake’s head.
  + Contour Detection: Contours representing potential positions of the snake's head are detected within the segmented image. These contours outline regions where the snake's head may be located, enabling precise localization.
  + Snake’s head Identification: Among the detected contours, the contour with the maximum area is identified as the snake's head. This contour likely corresponds to the largest object within the image, which is assumed to be the snake's head.
  + Movement Direction Update: The movement direction of the snake is updated based on the difference between the current and previous positions of its head. This ensures that the snake moves smoothly and responsively in the desired direction.
  + Collision detection: Collision detection is performed to check for collisions between the snake's head and obstacles or its own body. This prevents the snake from intersecting with obstacles or itself, which would result in game over.
  + Game Over/ Win Condition Check: If the snake collides with an obstacle or its own body, the game ends, and the player is prompted to restart. Additionally, the game checks for win conditions, where the player reaches a predefined score threshold, leading to victory.
  + Score and Snake Length Update: If the snake consumes food, its length increases, and the player's score increments. This rewards the player for successfully navigating the snake to consume food items.
  + Display Update: The game continuously updates the display with the current frame, snake, food, obstacles, and score. This ensures that the player receives real-time feedback on the game state and their progress.
  + Loop Continuation: The main loop continues iterating until the player chooses to exit the game. This allows for uninterrupted gameplay and provides the player with the option to play as long as desired.

### Reset Functionality:

Upon restarting the game, all variables are reset to their initial states, and new obstacle and food locations are generated.

## Code Structure