

# MINOR PROJECT REPORT

NAME – AMAAN KHAN

SECTION - 2CSE14

ROLL NO. – CS-23411442

SUBJECT – DEEP LEARNING

# SUBMITTED TO – MS. SURABHI PURWAR

**Index**

1. Introduction
2. Objective
3. Libraries and Tools Used
4. Detailed Explanation of Code
   * 4.1 Initialization
   * 4.2 Fruit Class
   * 4.3 Hand Detection
   * 4.4 Level Screen
   * 4.5 Main Game Loop
5. Functions Used
6. Challenges and Limitations
7. Possible Improvements
8. Conclusion

**1. Introduction**

This project recreates a simple version of the popular game "Fruit Ninja," but instead of using a mouse or touchscreen, it uses **hand detection through a webcam**. The player waves their hand in front of the camera to "cut" fruits that appear on the screen. The game includes sound effects, a score counter, levels, and a bomb penalty.

**2. Objective**

The goal of this project is to integrate **Computer Vision** and **Game Development** by using **Mediapipe** for hand tracking and **Pygame** for game display and interaction. This project demonstrates real-time object interaction using hand gestures without physical contact with the device.

**3. Libraries and Tools Used**

* **OpenCV (cv2):** Used to access webcam video, process video frames, and handle color conversions.
* **NumPy:** Supports numerical operations (minor role in this project).
* **Pygame:** Main library to create the game, handle screen updates, sounds, images, and collision detection.
* **Mediapipe:** Google's framework for real-time hand tracking and detection.
* **Random:** Used to randomly spawn fruits and bombs.
* **Time:** Used for timer management in the game.

**4. Detailed Explanation of Code**

**4.1 Initialization**

* Pygame and Pygame mixer are initialized.
* The display window size is set (1024x768).
* Background music and sound effects are loaded (with error handling).
* Images for fruits and bombs are loaded (or fallback circles are drawn if not available).

**4.2 Fruit Class**

The Fruit class manages all properties of the fruits and bombs.

Attributes:

* self.image: Fruit or bomb image.
* self.is\_bomb: Boolean flag to determine if it's a bomb.
* self.x, self.y: Random spawn position.
* self.speed: Falling speed.
* self.rect: Rectangle used for collision detection.
* self.cut: Status flag if the fruit is already cut.

Methods:

* move(): Updates the fruit's y-position.
* draw(): Renders the fruit image (or colored circle) onto the screen.

**4.3 Hand Detection**

* Using Mediapipe's Hands module.
* Converts BGR camera frames to RGB.
* Detects landmarks on the hand.
* Draws landmarks and creates a bounding box (hand\_rect) that is used for collision with fruits.

Function:

* detect\_hand(frame): Returns processed frame and hand bounding box.

**4.4 Level Screen**

Before each level, the function level\_screen(level) is called:

* Displays "Level X" text.
* Waits for 2 seconds before starting gameplay.

**4.5 Main Game Loop**

Function: game\_loop()

* Starts webcam feed.
* Continuously captures video frames.
* Displays the video feed on the game window.
* Spawns fruits randomly.
* Detects if the hand's bounding box overlaps with a fruit's rectangle.
* Updates the score: +1 for fruit, -1 for bomb.
* Moves fruits upward.
* Displays timer and score.
* Handles level transitions and game over conditions.
* Maintains 60 frames per second using Pygame's clock.

**5. Functions Used**

| **Function Name** | **Purpose** |
| --- | --- |
| pygame.init() | Initializes all imported pygame modules. |
| pygame.mixer.init() | Initializes the mixer module for sounds. |
| pygame.display.set\_mode() | Sets the window size for the game. |
| pygame.event.get() | Captures events like quit. |
| cv2.VideoCapture(0) | Opens webcam feed. |
| cv2.cvtColor() | Changes frame from BGR to RGB. |
| pygame.surfarray.make\_surface() | Converts camera frames to Pygame surface. |
| pygame.draw.circle() | Draws fallback colored circles. |
| pygame.Rect() | Creates rectangles for collision detection. |
| random.randint() | Randomly spawns fruits and determines speed. |
| time.time() | Tracks time for game duration. |
| pygame.quit() | Gracefully quits the game. |

**6. Challenges and Limitations**

* **Accuracy of hand detection:** Mediapipe is powerful, but lighting conditions or background clutter can cause misdetection.
* **Performance:** Real-time hand tracking and game rendering can slow down systems without GPUs.
* **No multiple hands:** Current implementation only tracks one hand.
* **Simple physics:** Fruits move linearly without gravity or bouncing effects.
* **Fallback mechanisms:** In case images or sounds are missing, basic colored circles and silent mode are used.

**7. Possible Improvements**

* Add **gravity-based physics** for more realistic fruit movement.
* Allow **multiple hands** or **multiple players**.
* Add **different fruit types** with bonus points.
* Add **animations** when fruits are cut.
* Include **high score saving** across sessions.
* Optimize **performance** for low-end machines.
* Add **better background visuals** instead of a white screen.
* Make a **mobile version** using Kivy or Pygame Mobile.

**8. Conclusion**

This project successfully demonstrates the integration of **Computer Vision** and **Game Development** in a simple, entertaining manner. By using **hand detection technology** instead of traditional controls, it showcases an emerging trend of touchless interaction. With some additional improvements, this game could be expanded into a full-fledged, gesture-controlled application. It is an excellent beginner to intermediate-level project for those interested in **AI**, **OpenCV**, **Pygame**, and **Interactive Media Applications**.

SNEAK PEAKS OF GAME



