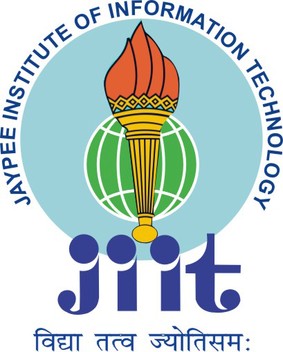
**Space Invaders**

**Open Source Software Lab Project**

**ODD SEM 2024**



# November – 2024

# Student’s name :

# Shivang Mishra 22103275 (B10)

# Sarthak Negi 22103288 (B10)

# Vanshika Agarwal 22103276 (B10)

# Prarabdh Agarwal 22103280 (B10)

**Submitted in partial fulfillment of the Degree of Bachelor of Technology in Computer Science Engineering**

# DEPARTMENT OF COMPUTER SCIENCE ENGINEERING & INFORMATION TECHNOLOGYJAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

**1. Introduction**

The Gesture-Controlled Space Invaders Game is an innovative project that integrates computer vision and game development to create an engaging, hands-free gaming experience. Inspired by the classic arcade game Space Invaders, this project replaces traditional input devices with hand gestures, allowing players to control a spaceship's movement and actions using a webcam.

This project demonstrates how modern technologies such as OpenCV, Mediapipe, and Pygame can work together to blend real-time gesture recognition with game mechanics, offering an immersive and interactive experience. It is designed to enhance accessibility, engagement, and creativity in gaming.

**2. Modeling and Implementation Details**

* **Game Description**

The game is a single-player arcade-style experience where the player controls a spaceship to destroy incoming enemies:

* **Player Actions**: Move left, move right, and fire bullets using specific hand gestures.
* **Enemies**: Six enemies move horizontally across the screen and advance downward when they hit screen edges. Colliding with the enemies or failing to shoot them before they reach the bottom ends the game.
* **Score Tracking**: A dynamic scoring system increments whenever an enemy is destroyed.
* **Gesture Recognition Mechanism**

Using Mediapipe, the game detects hand gestures through a live webcam feed:

* **Gestures and Actions:**
* **Left Movement**: Thumb positioned to the left of the index finger.
* **Right Movement**: Thumb positioned to the right of the index finger.
* **Shoot**: Thumb raised higher than all fingers, with middle, ring, and pinky fingers positioned below the index finger.

The gestures are processed in real-time and smoothed to ensure responsiveness while avoiding erratic inputs.

* **Game Loop**

The game operates in a continuous loop where:

* Input is captured via the webcam for gesture recognition.
* Game state is updated, including player movement, bullet dynamics, enemy positioning, and score updates.
* Collisions are detected between bullets and enemies.
* Graphics are rendered, including the game background, player, enemies, bullet, and score.
* Webcam feed is displayed alongside the game to provide feedback on gesture recognition.
* **Collision Detection**:

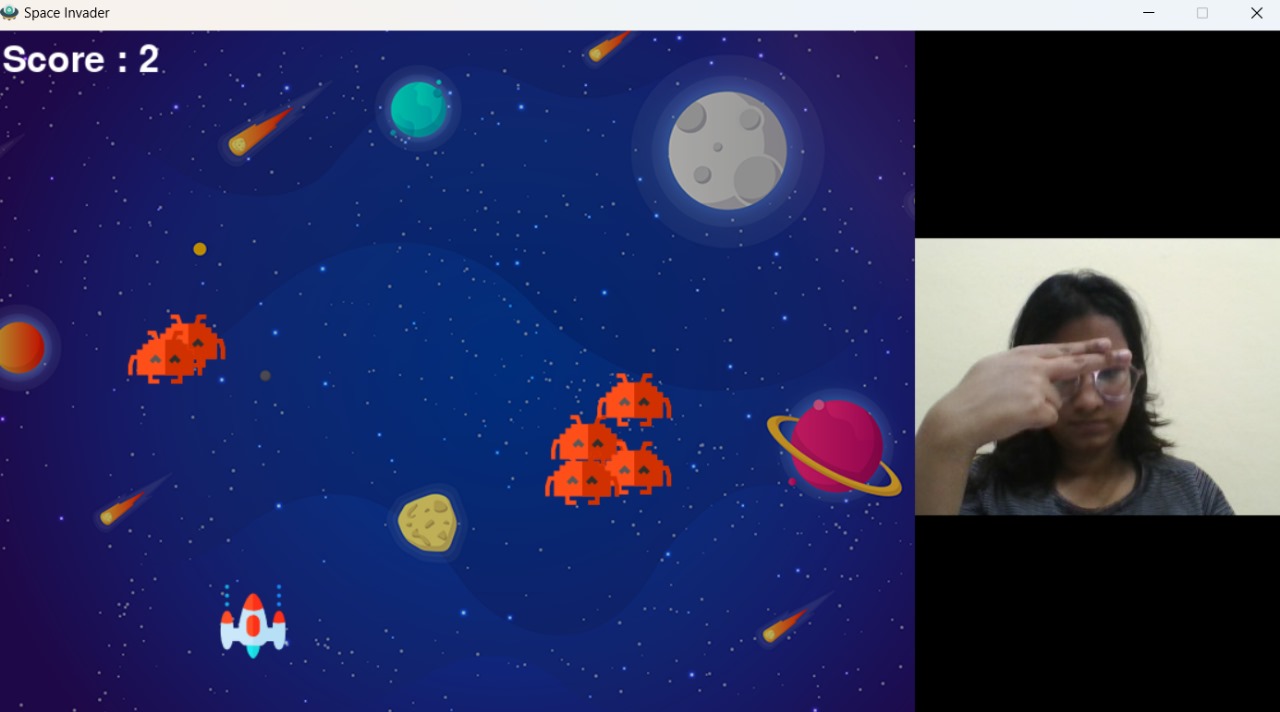
Enemy and bullet interactions are detected using the distance formula:

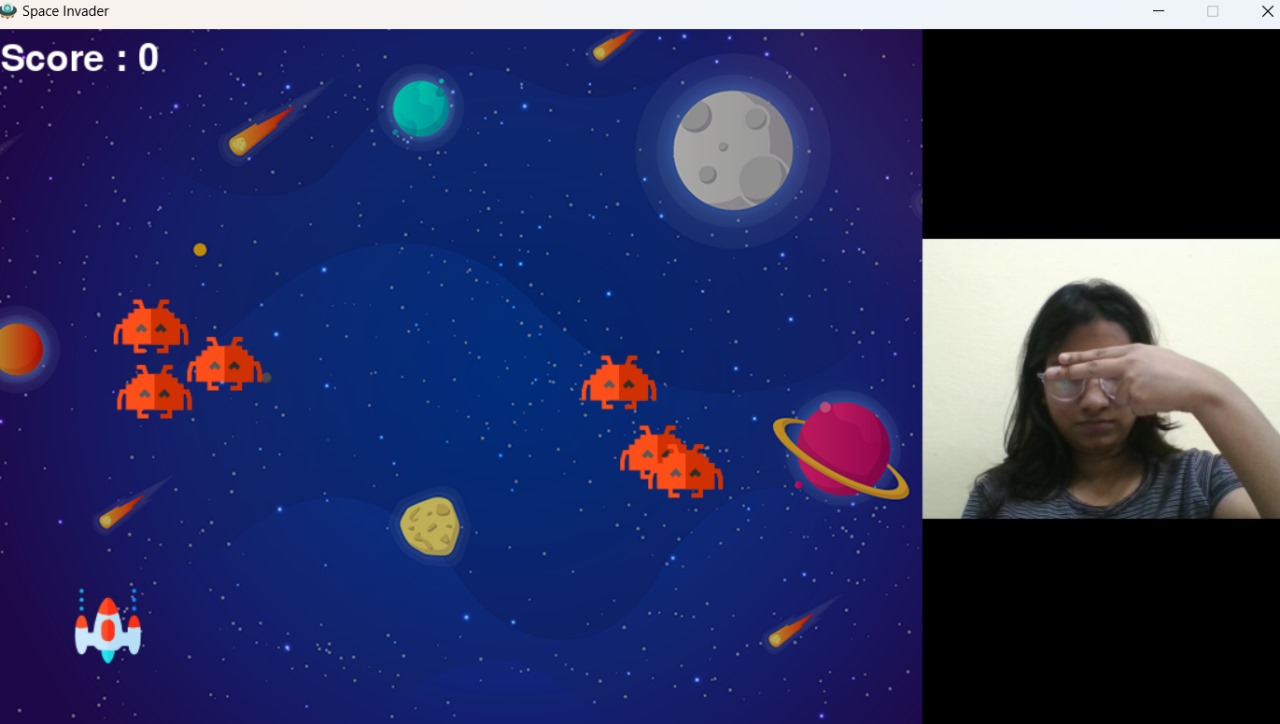
Distance =

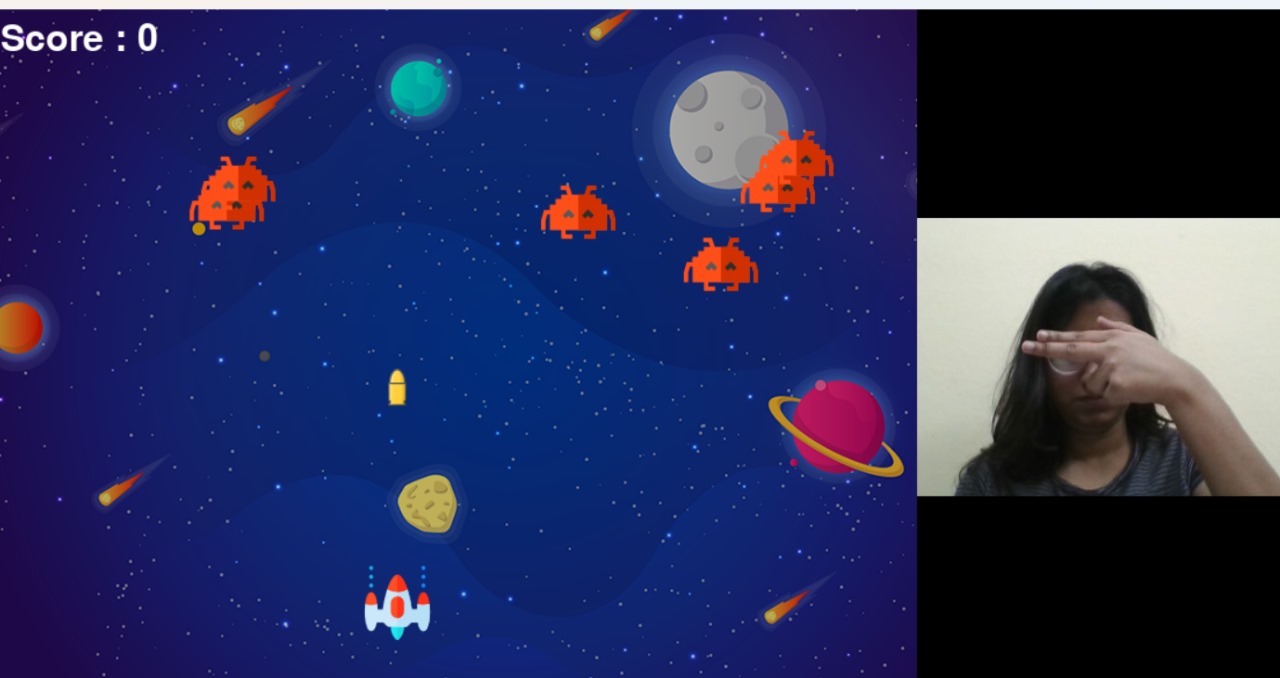


A collision is registered if the distance is less than 27 pixels.

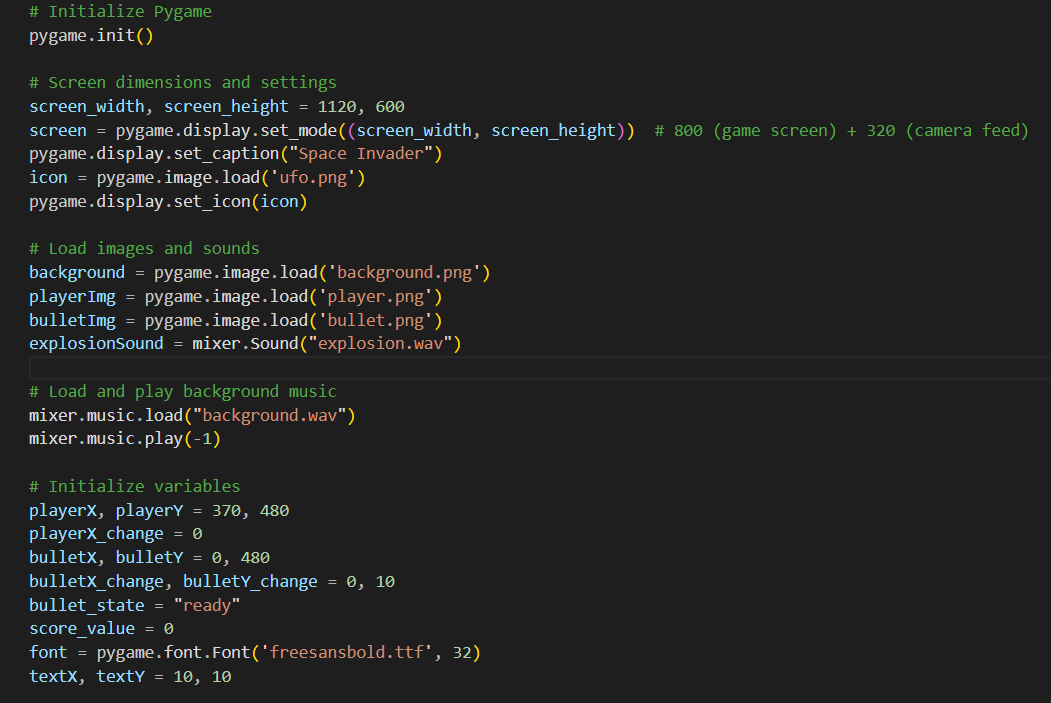
* **Snapshots of the Game:**







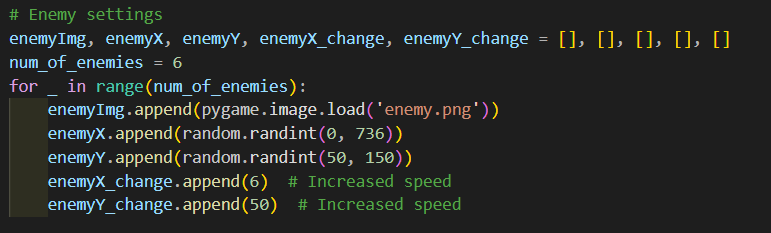
**3. Code snippets and working function.**



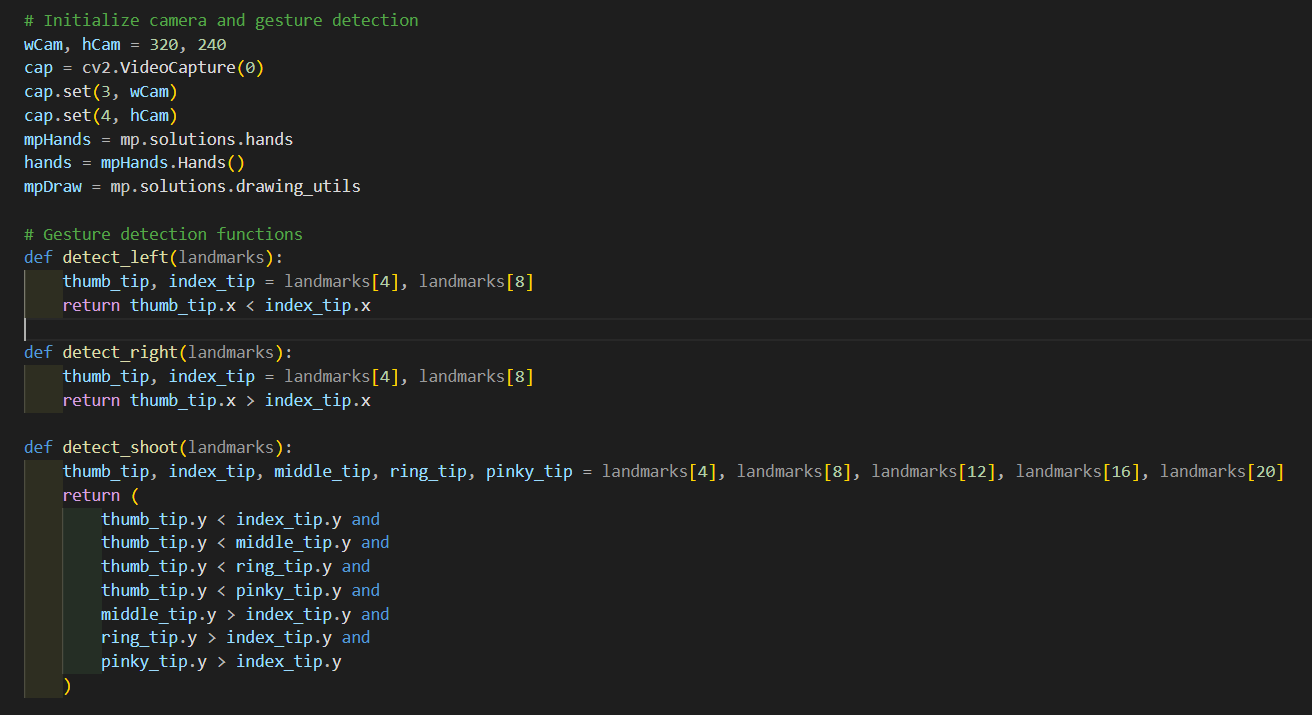
* **pygame.init()**  
  Initializes all Pygame modules to prepare them for use, setting up the environment for creating the game.
* **Screen dimensions and settings**
* screen\_width, screen\_height = 1120, 600 defines the size of the game window, with a width of 1120 pixels and height of 600 pixels.
* pygame.display.set\_mode((screen\_width, screen\_height)) creates the game window with the specified dimensions.
* pygame.display.set\_caption("Space Invader") sets the title of the game window to "Space Invader".
* pygame.image.load('ufo.png') loads an image to be used as the game window icon.
* pygame.display.set\_icon(icon) assigns the loaded image (icon) as the window's icon.
* **Initialize player variables**
* playerX, playerY = 370, 480 sets the initial coordinates of the player sprite, centering it horizontally and positioning it near the bottom of the screen.
* playerX\_change = 0 initializes the horizontal movement of the player to stationary.

 **Initialize bullet variables**

* bulletX, bulletY = 0, 480 sets the bullet's starting position. It begins aligned with the player vertically but hasn't been fired yet (x is arbitrary).
* bulletX\_change, bulletY\_change = 0, 10 defines the movement speed of the bullet. It only moves vertically, 10 pixels per frame.
* bullet\_state = "ready" initializes the bullet as inactive, meaning it isn't currently visible or moving on screen.
* **Initialize score variables**
* score\_value = 0 sets the player's score to 0 at the start of the game.
* font = pygame.font.Font('freesansbold.ttf', 32) specifies the font and size for displaying text such as the score.
* textX, textY = 10, 10 determines the position where the score text will appear, near the top-left corner of the screen.



* **enemyImg.append(pygame.image.load('enemy.png'))**  
  Loads the enemy sprite (enemy.png) and appends it to the enemyImg list. This allows each enemy to have the same appearance.
* **enemyX.append(random.randint(0, 736))**  
  Randomly assigns an initial horizontal position for each enemy within the range 0 to 736 pixels. This ensures that enemies start at different positions across the screen width (assuming a screen width of 800 and an enemy sprite width of 64 pixels).
* **enemyY.append(random.randint(50, 150))**  
  Randomly assigns an initial vertical position for each enemy within the range 50 to 150 pixels. This positions enemies at different heights near the top of the screen.
* **enemyX\_change.append(6)**  
  Sets the horizontal movement speed for each enemy. In this case, enemies move at a speed of 6 pixels per frame.
* **enemyY\_change.append(50)**  
  Specifies the vertical position shift when the enemy changes direction or moves down (e.g., after hitting a screen edge).



* **mpHands = mp.solutions.hands**  
  Imports the hands module from MediaPipe, a library designed for efficient hand-tracking and gesture recognition.
* **hands = mpHands.Hands()**  
  Creates an instance of the Hands class, which is used for detecting and tracking hand landmarks in the video feed.
* **mpDraw = mp.solutions.drawing\_utils**  
  Imports MediaPipe's drawing utilities for visualizing hand landmarks by drawing lines and points on the camera feed.

**detect\_left(landmarks)**

* **Purpose**: Detects if the thumb tip is positioned to the left of the index finger tip.
* **Logic**: Retrieves the x-coordinates of the thumb tip (landmarks[4]) and the index finger tip (landmarks[8]).

Returns True if the thumb is to the left (thumb\_tip.x < index\_tip.x), otherwise False.

**detect\_right(landmarks)**

* **Purpose**: Detects if the thumb tip is positioned to the right of the index finger tip.
* **Logic**: Retrieves the x-coordinates of the thumb tip (landmarks[4]) and the index finger tip (landmarks[8]).

Returns True if the thumb is to the right (thumb\_tip.x > index\_tip.x), otherwise False.

**detect\_shoot(landmarks)**

* **Purpose**: Detects a "shooting" gesture based on the relative positions of the fingers.
* **Logic**: Retrieves the y-coordinates of the thumb, index, middle, ring, and pinky tips.

Checks that the thumb tip is higher (y coordinate is smaller) than all the other fingers.

Ensures that the middle, ring, and pinky tips are below the index finger tip (middle\_tip.y > index\_tip.y, etc.).

Returns True if all conditions match the gesture.

**3. Technologies/Tools Used**

**1. Programming Language**

* Python: Primary language used for implementing the game logic, gesture recognition, and multimedia integration.

**2. Game Development Framework**

* Pygame:
  + Provides tools for creating 2D graphics.
  + Manages the game window, sprites, animations, and sound integration.

**3. Computer Vision Library**

* OpenCV:
  + Captures real-time video feed from the webcam.
  + Converts the video feed into a format suitable for gesture analysis.

**4. Gesture Recognition Framework**

* Mediapipe:
  + Detects and tracks hand landmarks in the video feed.
  + Processes gesture-specific logic for interpreting player commands.

**5. Audio Integration**

* Pygame Mixer:
  + Plays background music throughout the game.
  + Triggers sound effects like explosions for better immersion.

**6. Random Module**

* Python’s random module:
  + Randomizes enemy positions and movement directions, ensuring a unique gaming experience every time.

**4. Conclusion**

The **Gesture-Controlled Space Invaders Game** is a creative and impactful project that highlights the potential of integrating computer vision and game development to offer unique, hands-free interactions. By replacing traditional controls with hand gestures, the project demonstrates an innovative approach to gaming, making it more accessible and engaging.

This project showcases how cutting-edge technologies like **Mediapipe** for gesture recognition and **Pygame** for game mechanics can come together to create a highly interactive and dynamic user experience. The real-time interpretation of gestures adds a layer of immersion, bridging the gap between physical movement and digital interaction. Furthermore, the game retains the essence of the classic Space Invaders while introducing a modern twist through intuitive controls.

**Key Contributions**

1. **Innovative Gameplay**:
   * The use of hand gestures for controlling game elements is a novel approach, removing the need for traditional input devices like keyboards or controllers.
   * This enhances accessibility, especially for users who may have physical limitations or prefer alternative interaction methods.
2. **Real-Time Responsiveness**:
   * The seamless processing of webcam input and the accurate interpretation of gestures ensure smooth gameplay without noticeable lags, providing a polished experience.
3. **Integration of Technology**:
   * The project highlights the power of integrating multiple technologies, such as computer vision, game design, and audio processing, into a cohesive system.
4. **Engagement**:
   * The inclusion of dynamic elements like enemy movements, collision detection, scoring, and sound effects creates an engaging and immersive gaming environment.

**5. Future Enhancements**

* **Improved Gesture Recognition**: Incorporate machine learning models for robust gesture detection across diverse environments and users.
* **Difficulty Progression:** Gradually increase enemy speed or introduce new enemy types as the score increases.
* **Multiplayer Mode**: Enable multi-hand tracking for a co-op gaming experience.
* **Mobile Integration**: Port the game to mobile devices, using their built-in cameras for gesture recognition.

The Gesture-Controlled Space Invaders Game is a proof of concept showcasing how modern technologies can revolutionize the way we interact with games, setting the stage for future innovations in interactive entertainment.

**6. References**

* Pygame documentation : <http://www.pygame.org/docs/>
* OpenCV documentation : https://docs.opencv.org/
* Numpy documentation : https://numpy.org/doc/
* Mediapipe documentation : https://developers.google.com/mediapipe
* **Pygame Mixer** (Audio Module of Pygame) documentation : https://www.pygame.org/docs/ref/mixer.html