

# 1.INTRODUCTION

## 1.1 Image Processing

Image processing is the field of study and application that deals with modifying and analyzing digital images using computer algorithms. The goal of image processing is to enhance the visual quality of images, extract useful information, and make images suitable for further analysis or interpretation.

## 1.2 OpenCV- Introduction

OpenCV, Open Source Computer Vision Library, is an open-source computer vision and machine learning software library. Originally developed by Intel, it is now maintained by a community of developers under the OpenCV Foundation.

**Opencv** is a huge open-source library for computer vision, machine learning, and image processing. Now, it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even the handwriting of a human.

## 1.3 Image Processing Using OpenCV

OpenCV (Open Source Computer Vision) is a powerful and widely-used library for image processing and computer vision tasks. It provides a comprehensive set of functions and tools that facilitate the development of applications dealing with images and videos.

While taking photographs is as simple as pressing a button, processing and improving those images sometimes takes more than a few lines of code. That's where image processing libraries like OpenCV come into play. OpenCV is a popular open-source package that covers a wide range of image processing and computer vision capabilities and methods. It supports multiple programming languages including Python, C++, and Java. OpenCV is highly tuned for real-time applications and has a wide range of capabilities.

## 1.4 How to Install OpenCV for Python on Windows?

To install OpenCV, one must have Python and PIP, preinstalled on their system. To check if your system already contains Python, go through the following instructions: Open the **Command line**(search for **cmd** in the Run dialog( + **R**). Now run the following command:

```
Python --version
```

If Python is already installed, it will generate a message with the Python version available.

C:\Windows\system32\cmd.exe

```
C:\Users\Abhinav Singh>python --version
Python 3.8.1
C:\Users\Abhinav Singh>
```

If Python is not present, go through [How to install Python on Windows?](#) and follow the instructions provided. **PIP** is a package management system used to install and manage software packages/libraries written in Python. These files are stored in a large “on-line repository” termed as Python Package Index (PyPI). To check if PIP is already installed on your system, just go to the command line and execute the following command:

```
pip -V
```

```
C:\Windows\system32\cmd.exe
C:\Users\Abhinav Singh\AppData\Local\Programs\Python\Python38-32>pip -V
pip 19.3.1 from c:\users\abhinav singh\appdata\local\programs\python\python38-32\lib\site-packages\pip (python 3.8)
C:\Users\Abhinav Singh\AppData\Local\Programs\Python\Python38-32>
```

### Downloading and Installing OpenCV:

OpenCV can be directly downloaded and installed with the use of pip (package manager). To install OpenCV, just go to the command-line and type the following command:

```
!pip install opencv-python
```

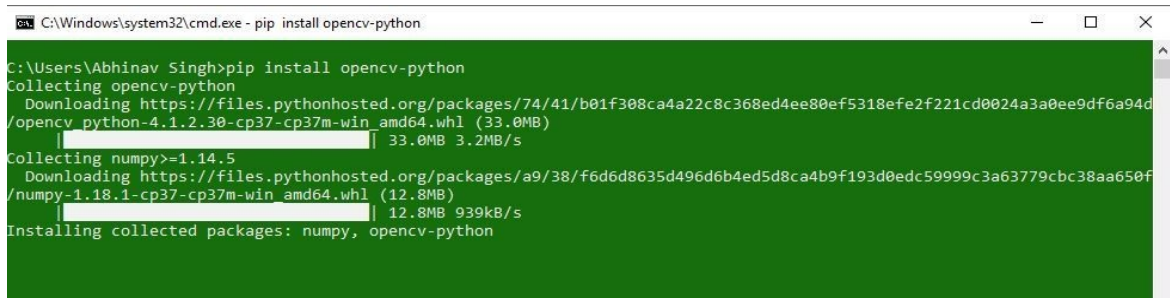
- **Type the command in the Terminal and proceed:**

```
C:\Windows\system32\cmd.exe - pip install opencv-python
C:\Users\Abhinav Singh>pip install opencv-python
```

- **Collecting Information and downloading data:**

```
C:\Windows\system32\cmd.exe - pip install opencv-python
C:\Users\Abhinav Singh>pip install opencv-python
Collecting opencv-python
  Downloading https://files.pythonhosted.org/packages/74/41/b01f308ca4a22c8c368ed4ee80ef5318efe2f221cd0024a3a0ee9df6a94d/opencv_python-4.1.2.30-cp37-cp37m-win_amd64.whl (33.0MB)
    | 6.5MB 226kB/s eta 0:01:58
```

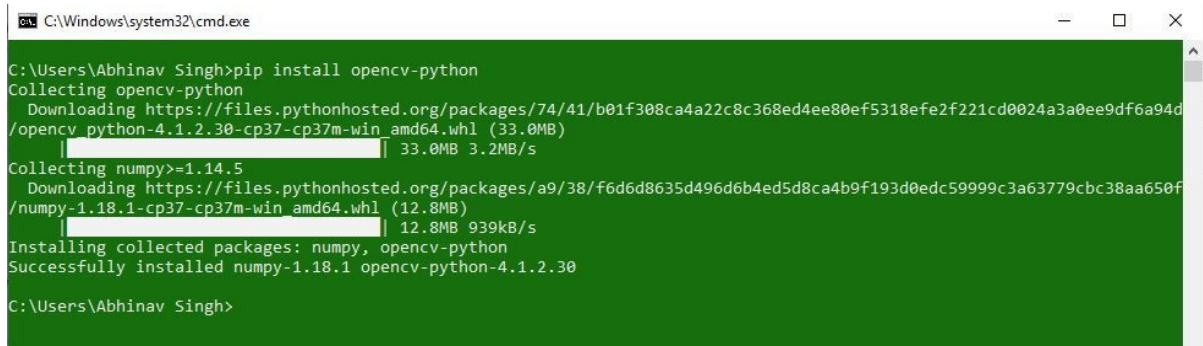
- **Installing Packages:**



```
C:\Windows\system32\cmd.exe - pip install opencv-python

C:\Users\Abhinav Singh>pip install opencv-python
Collecting opencv-python
  Downloading https://files.pythonhosted.org/packages/74/41/b01f308ca4a22c8c368ed4ee80ef5318efe2f221cd0024a3a0ee9df6a94d
/opencv_python-4.1.2.30-cp37-cp37m-win_amd64.whl (33.0MB)
    | 33.0MB 3.2MB/s
Collecting numpy>=1.14.5
  Downloading https://files.pythonhosted.org/packages/a9/38/f6d6d8635d496d6b4ed5d8ca4b9f193d0edc59999c3a63779cbc38aa650f
/numpy-1.18.1-cp37-cp37m-win_amd64.whl (12.8MB)
    | 12.8MB 939kB/s
Installing collected packages: numpy, opencv-python
```

- **Finished Installation:**



```
C:\Windows\system32\cmd.exe

C:\Users\Abhinav Singh>pip install opencv-python
Collecting opencv-python
  Downloading https://files.pythonhosted.org/packages/74/41/b01f308ca4a22c8c368ed4ee80ef5318efe2f221cd0024a3a0ee9df6a94d
/opencv_python-4.1.2.30-cp37-cp37m-win_amd64.whl (33.0MB)
    | 33.0MB 3.2MB/s
Collecting numpy>=1.14.5
  Downloading https://files.pythonhosted.org/packages/a9/38/f6d6d8635d496d6b4ed5d8ca4b9f193d0edc59999c3a63779cbc38aa650f
/numpy-1.18.1-cp37-cp37m-win_amd64.whl (12.8MB)
    | 12.8MB 939kB/s
Installing collected packages: numpy, opencv-python
Successfully installed numpy-1.18.1 opencv-python-4.1.2.30

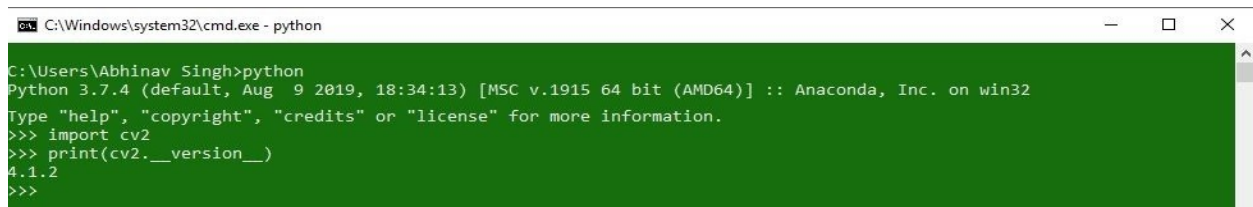
C:\Users\Abhinav Singh>
```

To check if OpenCV is correctly installed, just run the following commands to perform a version check:

```
python
```

```
>>>import cv2
```

```
>>>print(cv2.__version__)
```



```
C:\Windows\system32\cmd.exe - python

C:\Users\Abhinav Singh>python
Python 3.7.4 (default, Aug 9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> print(cv2.__version__)
4.1.2
>>>
```

## 1.5 Image Processing with Python

The following operations are most commonly uses for data augmentation task which training the model in Computer Vision.

- Image reading
- Image Resizing
- Image Rotation
- Image Translation

## Image reading

Following types of files are supported in OpenCV library:

- Windows bitmaps – \*.bmp, \*.dib
- JPEG files – \*.jpeg, \*.jpg
- Portable Network Graphics – \*.png
- WebP – \*.webp
- Sun rasters – \*.sr, \*.ras
- TIFF files – \*.tiff, \*.tif
- Raster and Vector geospatial data supported by GDAL

The steps to read and display an image in OpenCV are:

1. Read an image using imread() function.
2. Create a GUI window and display image using imshow() function.
3. Use function waitkey(0) to hold the image window on the screen by the specified number of seconds, 0 means till the user closes it, it will hold GUI window on the screen.
4. Delete image window from the memory after displaying using destroyAllWindows() function.

To read the images cv2.imread() method is used. This method loads an image from the specified file. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format) then this method returns an empty matrix.

**Syntax:** *cv2.imread(path, flag)*

**Parameters:**

**path:** A string representing the path of the image to be read.

**flag:** It specifies the way in which image should be read. It's default value is **cv2.IMREAD\_COLOR**

**Return Value:** This method returns an image that is loaded from the specified file.

**cv2.imshow()** method is used to display an image in a window. The window automatically fits the image size.

**Syntax:** *cv2.imshow(window\_name, image)*

**Parameters:**

**window\_name:** A string representing the name of the window in which image to be displayed.

**image:** It is the image that is to be displayed.

**Return Value:** It doesn't return anything.

**cv2.imwrite()** method is used to save an image to any storage device. This will save the image according to the specified format in current working directory.

**Syntax:** `cv2.imwrite(filename, image)`

**Parameters:**

**filename:** A string representing the file name. The filename must include image format like `.jpg`, `.png`, etc.

**image:** It is the image that is to be saved.

**Return Value:** It returns true if image is saved successfully.

## Image Resizing

Scaling operations increase or reduce the size of an image.

- **The cv2.resize() function** is used to resize an python image in OpenCV. It takes the following arguments:

```
cv2.resize(src, dsize, interpolation)
```

Here,

`src` :The image to be resized.

`dsize` :The desired width and height of the resized image.

`interpolation`:The interpolation method to be used.

## Image Rotation

Images can be rotated to any degree clockwise or otherwise. We just need to define rotation matrix listing rotation point, degree of rotation and the scaling factor.

- The **cv2.getRotationMatrix2D()** function is used to create a rotation matrix for an image. It takes the following arguments:
  - o The center of rotation for the image.
  - o The angle of rotation in degrees.
  - o The scale factor.
- The **cv2.warpAffine()** function is used to apply a transformation matrix to an image. It takes the following arguments:
  - o The python image to be transformed.

- o The transformation matrix.
  - o The output image size.
- The rotation angle can be positive or negative. A positive angle rotates the image clockwise, while a negative angle rotates the image counterclockwise.
- The scale factor can be used to scale the image up or down. A scale factor of 1 will keep the image the same size, while a scale factor of 2 will double the size of the python image.

## Image Translation

Translating an image means shifting it within a given frame of reference that can be along the x-axis and y-axis.

- **To translate an image using OpenCV**, we need to create a transformation matrix. This matrix is a  $2 \times 3$  matrix that specifies the amount of translation in each direction.
- **The cv2.warpAffine() function** is used to apply a transformation matrix to an image. It takes the following arguments:
  - o The image to be transformed.
  - o The transformation matrix.
  - o The output image size.
- **The translation parameters** are specified in the transformation matrix as the tx and ty elements. The tx element specifies the amount of translation in the x-axis, while the ty element specifies the amount of translation in the y-axis.

## 2. SYSTEM REQUIREMENTS

### 2.1 Software requirements

#### **Requirements for OpenCV and Anaconda**

- 32- or a 64-bit computer.
- For Anaconda—A minimum 3 GB disk space to download and install.
- Windows
- Python 2.7, 3.4, 3.5 or 3.6.

### 2.2 Hardware requirements

The hardware requirements are very minimal and the software can be made to run on most of the machines.

- Processor: Above X86
- Processor speed: 500 MHz and above.
- RAM: 64 MB or above storage space 8GB and more.

## 3. ABOUT THE PROJECT

### 3.1 Introduction to the project

The Aircraft War Game is a mini project designed to demonstrate the application of image processing techniques using OpenCV and Python in an interactive and engaging manner. The project simulates an aerial combat scenario where players control an aircraft to shoot down enemy targets while navigating through the game environment. This project combines elements of game development with advanced image processing, providing a practical example of how these technologies can be integrated to create a real-time interactive application.

#### Project Objectives

- **Demonstrate Image Processing Techniques:** Showcase how image processing techniques can be applied in a gaming context. This includes object detection, collision detection, and real-time visual effects.
- **Integrate OpenCV with Python:** Utilize OpenCV to manage game elements, detect and track objects, and handle user inputs within the Python programming environment.
- **Create an Engaging Game:** Develop a fun and interactive game that allows players to control an aircraft, engage in combat, and achieve high scores.

#### Project Design

##### 1. Game Mechanics:

- **Player Control:** Players control an aircraft using keyboard inputs. The aircraft can move across the screen and fire projectiles at enemy targets.
- **Enemies:** The game features various enemy aircraft that move across the screen, challenging the player to shoot them down while avoiding obstacles.
- **Scoring System:** Players earn points for successfully hitting enemy targets. The game keeps track of the score and displays it to the player.

##### 2. Image Processing Features:



- **Object Detection:** OpenCV algorithms are used to detect and track the player's aircraft and enemy targets. Functions like `cv2.findContours()` and `cv2.HoughCircles()` are employed to identify and follow these objects.
- **Collision Detection:** The game uses image processing techniques to determine when projectiles hit enemy targets or obstacles. This involves analyzing bounding boxes or pixel-level overlaps to detect collisions.
- **Visual Effects:** Various visual effects, such as explosions and power-ups, are implemented using image processing techniques to enhance the gaming experience. This includes real-time rendering of effects and transitions.

### 3. Implementation:

- **Image Loading and Display:** Images for the aircraft, enemies, and backgrounds are loaded and displayed using OpenCV's image handling functions.
- **Game Loop:** A main game loop continuously updates the game state, processes user inputs, and renders the game visuals. The loop ensures real-time interaction and smooth gameplay.
- **User Interface:** Basic user interface elements, such as score display and game messages, are created using OpenCV's GUI functions.

### 3.2 User-Defined Functions

User-defined functions are crucial for organizing and modularizing the code in the Aircraft War Game. These functions encapsulate specific functionalities related to game mechanics, image processing, and user interactions, making the code more readable, maintainable, and reusable. Below is an overview of some key user-defined functions that could be implemented in this project.

- **Function for Loading Game Assets**

```
import cv2

def load_image(file_path):
    image = cv2.imread(file_path)
    if image is None:
        raise FileNotFoundError(f"Image file {file_path} not found.")
```

```
return image
```

- **Function for Drawing Objects**

```
def draw_image(canvas, image, position):  
    x, y = position  
    h, w = image.shape[:2]  
    canvas[y:y+h, x:x+w] = image  
    return canvas
```

- **Function for Detecting Collisions**

```
def detect_collision(obj1_pos, obj1_size, obj2_pos, obj2_size):  
    x1, y1 = obj1_pos  
    w1, h1 = obj1_size  
    x2, y2 = obj2_pos  
    w2, h2 = obj2_size  
    return not (x1 + w1 < x2 or x1 > x2 + w2 or y1 + h1 < y2 or y1 > y2 + h2)
```

- **Function for Moving Objects**

```
def move_object(position, velocity):  
    x, y = position  
    vx, vy = velocity  
    new_position = (x + vx, y + vy)  
    return new_position
```

- **Function for Handling User Input**

```
import pygame  
def handle_input():  
    actions = {'move_up': False, 'move_down': False, 'move_left': False, 'move_right': False,  
    'fire': False}  
    for event in pygame.event.get():
```

```
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_UP:
        actions['move_up'] = True
    elif event.key == pygame.K_DOWN:
        actions['move_down'] = True
    elif event.key == pygame.K_LEFT:
        actions['move_left'] = True
    elif event.key == pygame.K_RIGHT:
        actions['move_right'] = True
    elif event.key == pygame.K_SPACE:
        actions['fire'] = True
elif event.type == pygame.KEYUP:
    if event.key == pygame.K_UP:
        actions['move_up'] = False
    elif event.key == pygame.K_DOWN:
        actions['move_down'] = False
    elif event.key == pygame.K_LEFT:
        actions['move_left'] = False
    elif event.key == pygame.K_RIGHT:
        actions['move_right'] = False
    elif event.key == pygame.K_SPACE:
        actions['fire'] = False
return actions
```

## 4. DESIGN

The design of an aircraft war game typically involves several key modules, each responsible for different aspects of the game's functionality. Below, I'll describe the main modules and their interactions, followed by a process/flows diagram.

### Key Modules

1. Asset Management Module
2. Game Engine Module
3. Rendering Module
4. Collision Detection Module
5. Input Handling Module
6. Physics and Movement Module
7. UI/Score Management Module

### Module Descriptions

#### 1. Asset Management Module

**Purpose:** Manages game assets such as images, sounds, and textures.

#### Responsibilities:

- Load and manage game resources.
- Provide assets to other modules as needed.

#### Key Functions:

- `load_image(file_path)`: Load images from disk.
- `load_sound(file_path)`: Load sound files.
- `get_asset(name)`: Retrieve an asset by name.

#### 2. Game Engine Module

**Purpose:** Coordinates the overall game flow and integrates various game modules.

#### Responsibilities:

- Initialize and run the main game loop.
- Update game states and call appropriate modules.
- Manage the game's lifecycle (start, pause, end).

**Key Functions:**

- `start_game()`: Begin the game and start the main loop.
- `update_game()`: Update game states and handle logic.
- `render_game()`: Call the rendering module to draw game objects.

### **3. Rendering Module**

**Purpose:** Handles drawing of game objects on the screen.

**Responsibilities:**

- Draw all visual elements, including aircraft, explosions, and backgrounds.
- Refresh the display to show updated game states.

**Key Functions:**

- `draw_image(canvas, image, position)`: Draw images onto the game canvas.
- `clear_screen()`: Clear the screen before drawing new frames.

### **4. Collision Detection Module**

**Purpose:** Detect collisions between game objects, such as aircraft and obstacles.

**Responsibilities:**

- Check if objects intersect or overlap.
- Trigger events or changes based on collision outcomes.

**Key Functions:**

- `detect_collision(obj1_pos, obj1_size, obj2_pos, obj2_size)`: Check if two objects collide.

### **5. Input Handling Module**

**Purpose:** Capture and process user inputs for controlling game actions.

**Responsibilities:**

- Handle keyboard, mouse, or controller inputs.
- Translate inputs into game actions (e.g., move aircraft, fire weapons).

**Key Functions:**

- `handle_input()`: Capture and return user inputs.
- `process_input(actions)`: Apply user inputs to game objects.

**6. Physics and Movement Module**

**Purpose:** Manage the physics and movement of game objects.

**Responsibilities:**

- Update the position and velocity of objects based on physics rules.
- Implement gravity, acceleration, and other physical effects.

**Key Functions:**

- `move_object(position, velocity)`: Update object position based on velocity.
- `apply_physics()`: Apply physics rules to game objects.

**7. UI/Score Management Module**

**Purpose:** Handle the user interface and score tracking.

**Responsibilities:**

- Display scores, health, and other game statistics.
- Manage UI elements such as menus and game over screens.

**Key Functions:**

- `update_score(new_score)`: Update the player's score.
- `display_ui()`: Render the user interface on the screen.

## Process/Flows Diagram

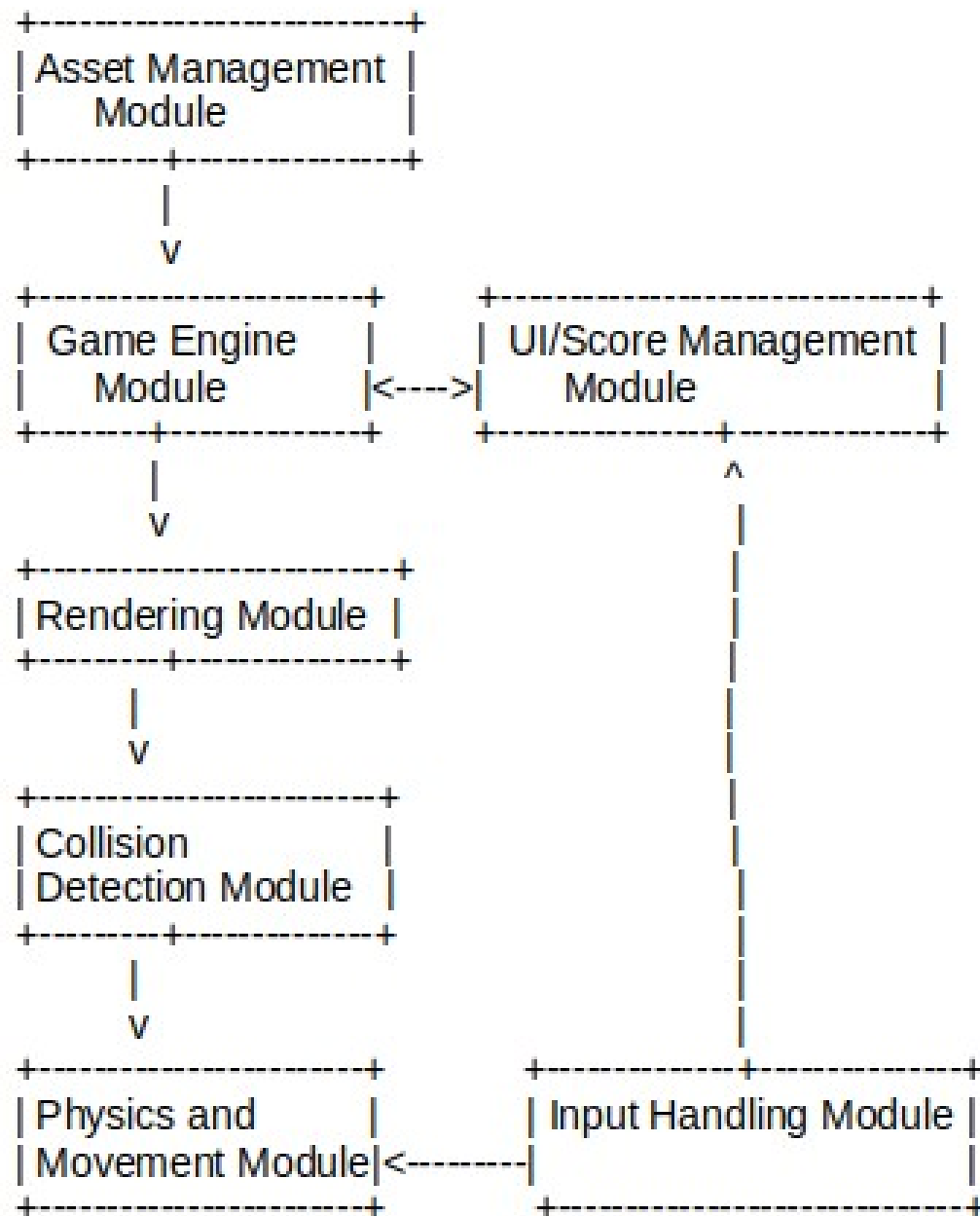


Fig :- Flows Diagram

## **Process Flow Explanation**

### **1. Initialization:**

- The Asset Management Module loads all necessary game assets.
- The Game Engine Module initializes and starts the game loop.

### **2. Game Loop:**

- **Input Handling:** The Input Handling Module captures user inputs and sends them to the Game Engine Module.
- **Physics and Movement:** The Game Engine Module updates the positions of objects using the Physics and Movement Module.
- **Collision Detection:** The Collision Detection Module checks for collisions and triggers events as needed.
- **Rendering:** The Rendering Module draws the updated game state on the screen.

### **3. UI Updates:**

- The UI/Score Management Module updates and displays game scores and other UI elements.

### **4. Game End:**

- The Game Engine Module manages game over conditions and transitions to end screens if necessary.



## 5. IMPLEMENTATION

The implementation phase of the **Aircraft War Game** involved translating the design and planning stages into a fully functional 2D shoot-'em-up game. This phase included setting up the development environment, coding the core game features, integrating assets, and ensuring a smooth gameplay experience.

### 1 Development Environment

The development environment for the Aircraft War Game was set up using Python and OpenCV. The necessary libraries were installed, and the project structure was organized to manage game assets, scripts, and resources efficiently.

### 2 Import Libraries

#### Testing and Debugging

##### Unit Testing

Individual functions were tested to ensure they worked correctly. This involved verifying asset loading, object rendering, and movement functionalities.

##### Integration Testing

The complete game was tested to ensure that all components worked together seamlessly. This included checking for proper object interactions and overall game stability.

##### Debugging

Debugging tools and techniques were used to identify and fix issues. Print statements and debugging tools were employed to trace and resolve any problems in the code.

##### Planning and Scheduling

The project was planned and executed over a 15-day period, divided into four key phases:

- **Analysis (2 days):** Understanding project requirements and setting objectives.

- **Design (5 days):** Creating the game design, including asset creation and game mechanics.
- **Implementation (7 days):** Coding the game features and integrating components.
- **Testing (1 day):** Conducting final tests and debugging.

### **Development Iterations**

- **Iteration 1:** Focused on project analysis and defining the scope.
- **Iteration 2:** Game design, including brainstorming and early decisions on requirements.
- **Iteration 3:** Coding and internal training, achieving milestones.
- **Iteration 4:** Finalizing the product and integrating all components.

### **Risk Management**

#### **Identified Risks and Mitigations**

- **Risk of Project Delay:** Managed by adhering to the project schedule and regular progress reviews.
- **User Acceptance:** Positive feedback anticipated due to engaging gameplay and simple controls.
- **Requirement Changes:** Low probability, as project scope was well-defined.
- **Marketing and Technology Fit:** Evaluated and addressed to ensure successful project delivery.
- **Quality and Decision-making:** Ensured through thorough planning and iterative development.

## 6. SNAPSHOTS

6.1 :-

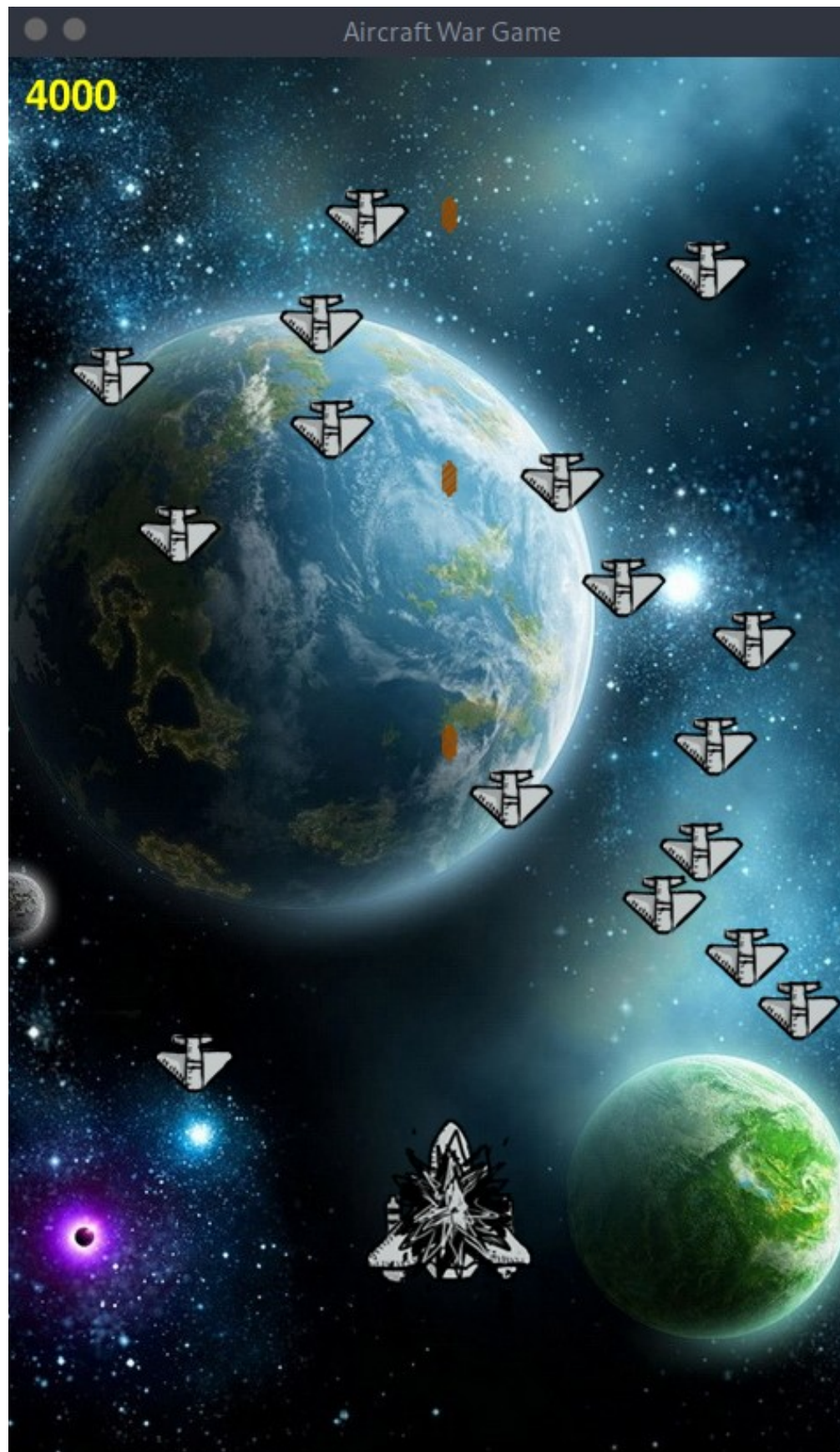


Fig :- Game AirCraft War has Start

6.2 :-

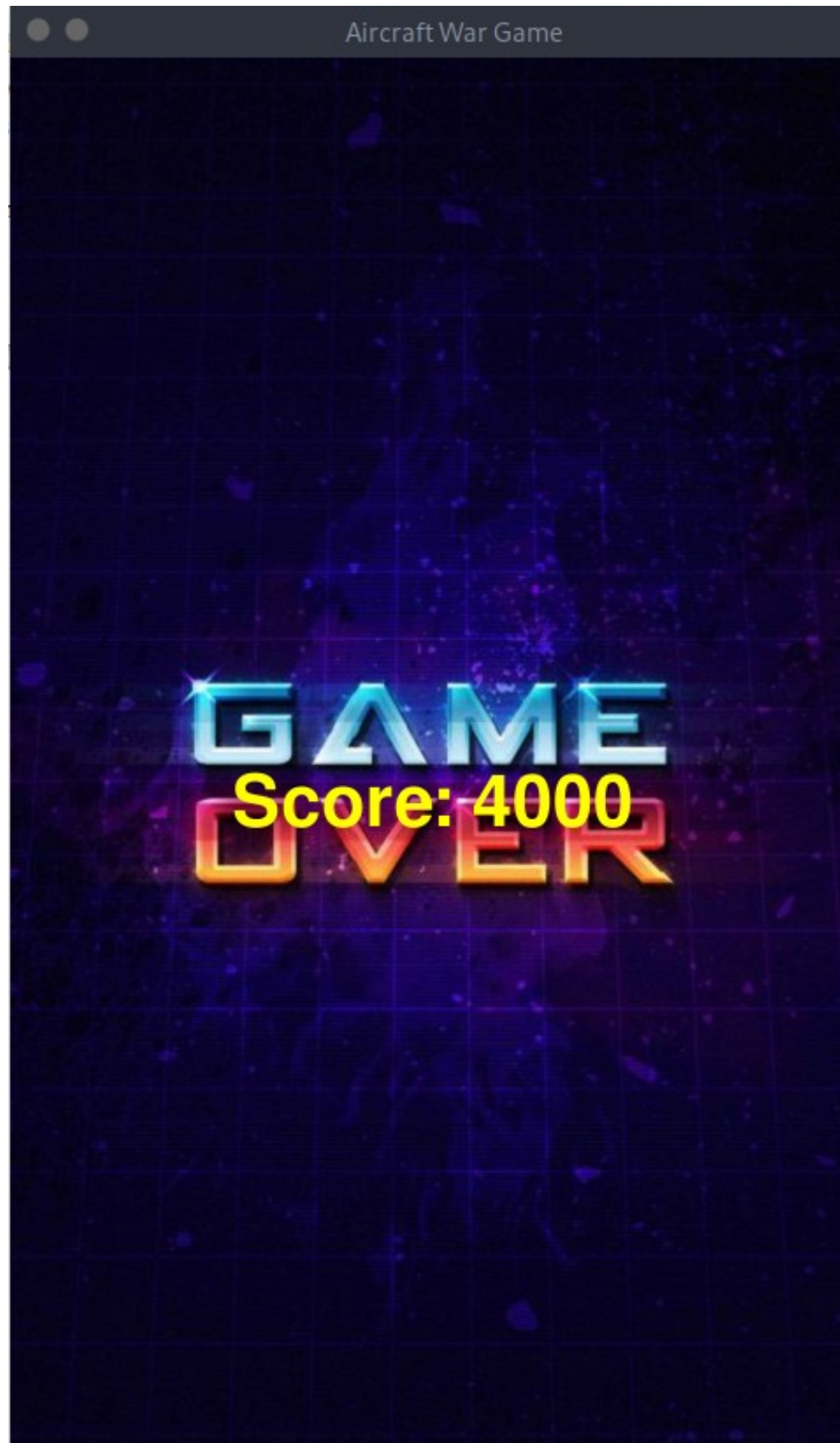


Fig :- AirCraft War has ended and Score

## 7. CONCLUSION

There has been a lot of explanation of what has been and what has not been implemented, followed by short discussions. The last part of the thesis is going to focus solely on the group's thoughts of the results, as well as our thoughts regarding further development.

Regarding the video-game itself, all of the game logic and the objects were designed and subsequently implemented using Unity and its 2D toolset. As a result, the video-game is composed of one single perfectly playable level which includes many different objects and obstacles to overcome. This is the part that I have most enjoyed, because I really like programming and designing game logic. Additionally, coming up with different Enemies and Boss logic provided me a fun challenge, and gave me the opportunity to learn many different issues about game design and development.

Furthermore, I discovered many interesting things about developing a game for people with colorblindness disability, and found out a way for them to thoughtfully enjoy the game. On the other hand, the obtained video-game can be played at different operating systems and it has a very good performance even on dated computers.

Moreover, it can be easily ported to any of the current generation game consoles.

Finally, I would like to highlight that most of the things done throughout this project were learned while developing it, as I had very little experience in the whole process of video-game development. I had made very simple video games without using full fledged game engines, and never before had I designed any sound effect or graphic material for them

### **(a) The Obstacles:**

1. Working with Visual Studio Code IDE for game development is completely a new experience for me. Normally i am working with different OO languages, DBMS, mark up languages etc.
2. It is very sensible work and it demands much time because the game engines try to connect game environment with the real world.
3. Creating a 2d model is very difficult because you need to work with each and every point of the model.
4. The Pygame module of python demands vast knowledge about its properties, functions, keywords and methods.

### **(b) The Achievements:**

Now, I know much more about game engines. How it works? The properties, objects and others.



## 8. BIBLIOGRAPHY

### Python Programming Language

- The Python programming language is fundamental for game development in this project. Its extensive libraries and simplicity make it a popular choice for creating interactive applications.
- Python Software Foundation. (n.d.). *Python Documentation*. Retrieved from <https://docs.python.org/>

### Visual Studio Code

- Visual Studio Code is the integrated development environment (IDE) used for coding the Aircraft War Game. The official documentation includes features, extensions, and best practices for effective coding.
- Microsoft. (n.d.). *Visual Studio Code Documentation*. Retrieved from <https://code.visualstudio.com/docs>

### Game Design Principles

- Various sources provide fundamental principles of game design, including mechanics, user experience, and engagement strategies. These principles guide the creation of compelling and enjoyable games.
- Schell, J. (2008). *The Art of Game Design: A Book of Lenses*. Morgan Kaufmann Publishers.

### 2D Game Development Techniques

- Books and online resources on 2D game development offer practical advice and techniques for creating engaging 2D games, including handling graphics, physics, and user input.
- Williams, R. (2013). *Game Development Essentials: An Introduction*. Delmar Cengage Learning.

### Computer Graphics Techniques

- Resources on computer graphics provide a deeper understanding of visual techniques and rendering, essential for creating high-quality game visuals.
- Hearn, D., & Baker, M. P. (2010). *Computer Graphics with OpenGL*. Pearson.