Game – N – Geek INTERACTIVE GAMINF PORTAL

Parth Borana [a], Vadarevu Sai Samprreet Khushal [b], Saimrika Gupta[c], Ashmit Gupta [d]

[a] *School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, Tamil Nadu, India* [abc]

Abstract

Most of the population, especially, elderly people are interested in computer gaming technology and desire to be actively involved in such activities, however they are limited by their responsiveness due to age, as gaming requires fast reflexes in order to provide and comfortable / satisfactory experience. People who are not very knowledgeable about gaming are also limited to enter the gaming world. This upbrings a huge demand in the current market for a software / product which can enable users who are new to the gaming world to be able to comfortably enter / transition into the gaming world or be able to enjoy the merits of gaming without their limitations stopping them.

1 INTRODUCTION

Game-N-Geek allows for interactive gaming experience which uses tracker to be able to interact in gaming environments using human movements.

Game-N-Geek appears to be a platform that utilizes tracking technology to allow for interactive gaming experiences that incorporate human movements. This technology can provide a more immersive and engaging gaming experience by allowing players to physically interact with the game environment using their body movements.

There are several types of tracking technology that can be used for this purpose, including motion capture systems, camera-based tracking, and wearable sensors. These technologies can detect and interpret a player's movements, allowing them to control their in-game avatar in real time.

Overall, the use of tracking technology in gaming has the potential to revolutionize the way we play and experience games. By incorporating human movements into the gameplay, we can create more immersive and interactive experiences that are both fun and engaging.

Game-N-Geek is a program that allows users to play any PC related video game using their hand signs and movements instead of using mouse and keyboard. It allows users to play games using hand signs.

Game-N-Geek allows for a smooth gaming experience without the inclusion of hands and can be used to interact within the gaming environment entirely in a visual manner..

2. Papers Used

2.1: Real Time Hand Gesture Movements Tracking and Recognizing System.

This paper proposes the use of contour detection, fingers movement detection, motion detection and segmentation, and applying these as a mouse functionality to be able to use the hand as a mouse which can interact with the given system

2.2: HCI Issues in Computer Games.

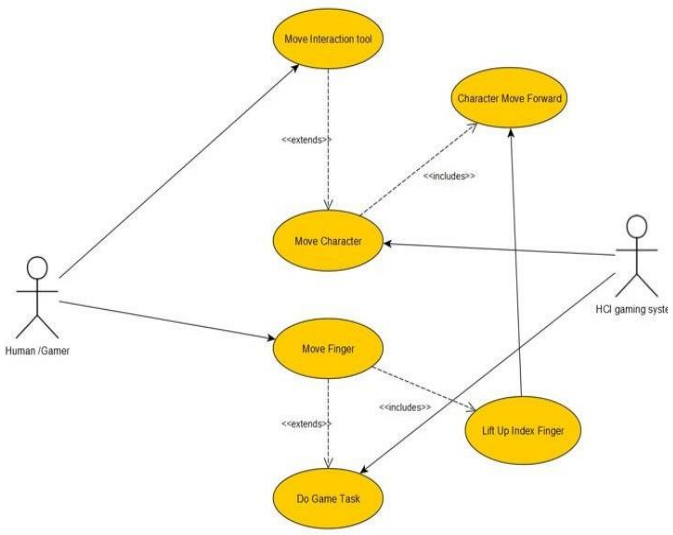
This paper discusses about the various issues which are brought up during HCI involvement in computer games, this includes factors such as: Theoretical, Methodological, Technical, and social Cultural aspects. It also brings to light the issue of the familiarity of many users with the QWERTY keyboard style which may hinder their performance/ Experience with the game when they have to use HCI related gadgets, gaming tools to perform the same activities.

2.3: Hand landmarks detection guide

You can identify the hands' landmarks in a picture using the Media Pipe Hand Land marker task. This task can be used to localize the hands' important areas and render visual effects on top of the hands. This task generates hand landmarks in image coordinates, hand landmarks in world coordinates, and handedness (left/right hand) of numerous detected hands using image data and a machine learning (ML) model as static data or a continuous stream.

3. REQUIREMENT ANALYSIS

SOFTWARE REQUIREMENTS



SOFTWARE REQUIREMENTS:

The software requirements for an interactive gaming experience that uses trackers to enable human movements can vary depending on the specific implementation of the technology. However, some general software requirements may include:

1.Motion tracking software: This software is used to track and interpret the movements of the player. Depending on the tracking technology used, this software may be provided by the manufacturer of the tracking system or may be developed in-house by the game developers.

2.Game engine: The game engine is the software that drives the game itself, providing the underlying framework for the gameplay mechanics, user interface, and graphics. Examples of popular game engines include Unity, Unreal Engine, and CryEngine.

3.User interface design tools: Designing an effective user interface for an interactive gaming experience that uses trackers can be complex. Game developers may use specialized software tools, such as Adobe XD or Sketch, to design and prototype user interfaces for the game.

4.Audio and visual software: To create a high-quality gaming experience, developers may use specialized software tools for audio and visual design. For example, Adobe Audition or Audacity can be used for audio editing, and Adobe Photoshop or GIMP can be used for image editing.

5.Network programming tools: If the interactive gaming experience involves multiplayer functionality, developers will need to use network programming tools to enable communication between players over the internet. Examples of such tools include Unity Multiplayer or Photon.

6.Analytics tools: To analyze player behavior and performance, developers may use specialized software tools for data analytics. Examples of such tools include Google Analytics or Mixpanel.

TOOLS USED:

- OpenCV (cv2): OpenCV is a computer vision library that offers several tools for processing images and videos. The given code makes use of OpenCV for image processing (color conversion, thresholding, contour detection), sketching on pictures, and video capture from a camera.

- A Python library for numerical computation is called NumPy (np). It is often used in the code while manipulating arrays and performing mathematical computations.

- Time: The Python time module offers a number of time-related functions. It is used in the code to measure the time it takes for certain actions to complete and to introduce delays.

- Mouse (m): The mouse module offers features for computer-based simulation of mouse events, such as clicks and movement. The mouse module is used in the given code to imitate mouse clicks.

- Using the Mediapipe library, the HandTrackingModule custom module allows for the detection and tracking of hands in pictures and videos.

- Python module PyAutoGUI (pg) is used to automate GUI interactions. The given code uses PyAutoGUI to mimic computer keyboard events, such as key presses.

- PyDirectInput can simulate keyboard and mouse inputs on a Windows PC. It is used to replicate keyboard events in the specified code. (e.g., pressing a key).

- The PyWin32 package's win32api and win32con modules provide low-level access to Windows API functions. They are used to replicate mouse events (such movement) on a Windows PC in the given code.

Pseudocode:

Import required libraries:

```
import cv2 as cv
import datetime
import numpy as np
import time
import mouse as m
import time
import HandTrackingModule as htm
import pyautogui as pg
import pydirectinput as pn
import win32api,win32con
```

3.2 Set up camera:

```
wCam, hCam = 1280, 720
plocx, plocy = 0, 0
clocx, clocy = 0, 0
cap = cv.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)
```

3.3 Define the finger tips to track:

```
tipIds = [4, 8, 12, 16, 20]
```

3.4 Initialize the hand detector:

```
last_click = datetime.datetime.now()
detector = htm.HandDetector(maxHands=1, detectionCon=0.75)
```

3.5 Loop over video frames and Read video frame:

```
while True:
success, img = cap.read()
  img = cv.flip(img, 1)
  hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
```

3.6 Read video frame:

```
success, img = cap.read()
  img = cv.flip(img, 1)
  hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
```

3.7 Define the range of red color to detect:

```
lower_red = np.array([0, 139, 184])
  upper_red = np.array([179, 255, 255])
  mask_red = cv.inRange(hsv, lower_red, upper_red)
```

3.8 Find contours of red objects:

```
contoursRed,_=cv.findContours(mask_red,
cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
  for c in contoursRed:
    # Ignore small contours
    if cv.contourArea(c) <= 50:
      continue
```

3.9 Get the bounding rectangle of the contour, Smooth the movement of the cursor:

```
x, y, _, _ = cv.boundingRect(c)
clocx = plocx + (x - plocx) / 6
    clocy = plocy + (y - plocy) / 6
```

3.10 Move the cursor:

win32api.mouse_event(win32con.MOUSEEVENTF_MOVE, int(clocx - plocx), int(clocy - plocy), 0, 0)

3.11 Detect hands in the frame:

```
 img = detector.findHands(img)

    lmList = detector.findPosition(img)
```

3.12 Check if hand is detected:

```
if len(lmList) != 0:

        fingers = []
```

3.13 Check if thumb is up:

```
 if lmList[tipIds[0]][1] > lmList[tipIds[0] - 1][1]:

            fingers.append(1)

        else:

            fingers.append(0)
```

3.14 Check if fingers are up:

```
for id in range(1, 5):

            if lmList[tipIds[id]][2] < lmList[tipIds[id] - 1][2]:

                fingers.append(1)

            else:

                fingers.append(0)
```

3.15 Control keyboard and mouse based on finger positions:

```
 if fingers[1] == 1:

            pn.keyDown('w')

        else:

            pn.keyUp('w')


        if fingers[0] == 1:

            m.press()

        else:

            m.release()
```

3.16 Display the image:

```
cv.imshow("Image", img)
```

3.17 Convert the image to the HSV color space:

```
hsv = cv.cvtColor(img,cv.COLOR_BGR2HSV)
```

3.18 Detect hands in the frame:

```
)
```

3.19 Define the lower and upper bounds of the red color to be detected:

```
 lower_red = np.array([0,139,184])

upper_red = np.array([179,255,255])
```

3.20 Create a mask using the specified color range:

```
mask_red = cv.inRange(hsv, lower_red, upper_red)
```

3.21 DeFind the contours of the objects in the red color range:

```
contoursRed,_=cv.findContours(mask_red, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

3.22 Determine if the thumb is up or down:

```
if lmList[tipIds[0]][1] > lmList[tipIds[0]-1][1]:

        fingers.append(1)

    else:

        fingers.append(0)
```

3.23 Determine if each finger is up or down:

```
for id in range(1,5):

        if lmList[tipIds[id]][2] < lmList[tipIds[id]-1][2]:

            fingers.append(1)

        else:

            fingers.append(0)
```

3.24 Press and hold the W key if the index finger is up:

```
 if fingers[1]==1:

        pn.keyDown('w')
```

```
    else:

        pn.keyUp('w')
```

3.25 Press and hold the left mouse button if the thumb is up:

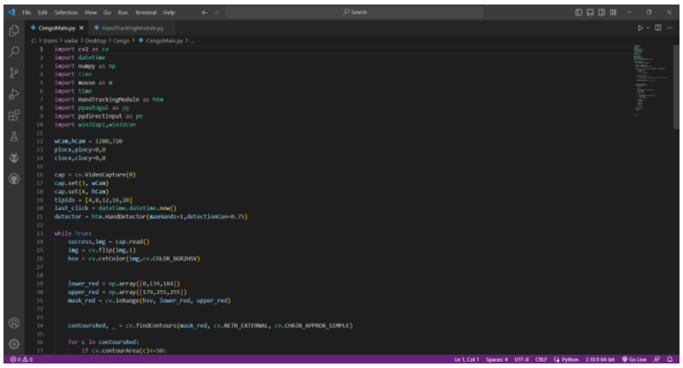
```
 if fingers[0]==1:

  m.press()

    else:

        m.release()
```

3.26 Check for key presses:

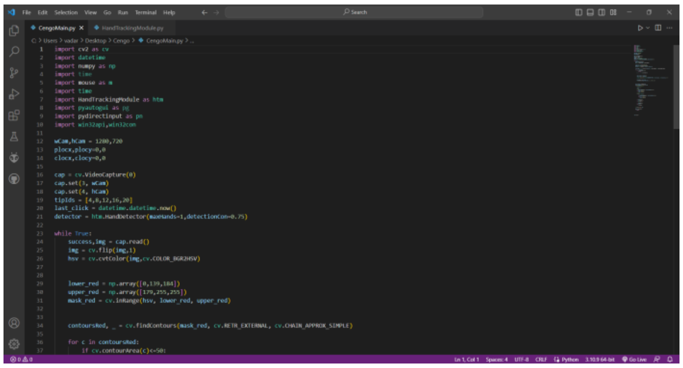
```
key = cv.waitKey(1)

if key==27:

    break;
```

3.27 Release the capture and close all windows:

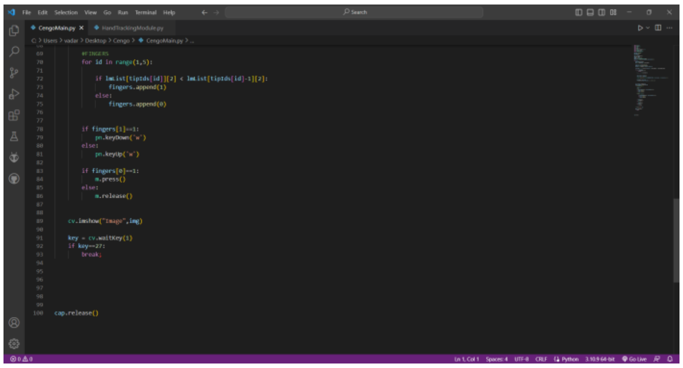
```
cap.release()

cv.destroyAllWindows()
```



```
import cv2 as cv
import datetime
import numpy as np
import time
import mouse as m
import time
import HandTrackingModule as htm
import pyautogui as pg
import pydirectinput as pn
import win32api,win32con

wCam,hCam = 1280,720
plocx,plocy=0,0
clocx,clocy=0,0

cap = cv.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)
tipIds = [4,8,12,16,20]
last_click = datetime.datetime.now()
detector = htm.HandDetector(maxHands=1,detectionCon=0.75)

while True:
    success,img = cap.read()
    img = cv.flip(img,1)
    hsv = cv.cvtColor(img,cv.COLOR_BGR2HSV)


    lower_red = np.array([0,139,184])
    upper_red = np.array([179,255,255])
    mask_red = cv.inRange(hsv, lower_red, upper_red)


    contoursRed, _ = cv.findContours(mask_red, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

    for c in contoursRed:
        if cv.contourArea(c)<=50:
```



```
import cv2 as cv
import datetime
import numpy as np
import time
import mouse as m
import time
import HandTrackingModule as htm
import pyautogui as pg
import pydirectinput as pn
import win32api,win32con

wCam,hCam = 1280,720
plocx,plocy=0,0
clocx,clocy=0,0

cap = cv.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)
tipIds = [4,8,12,16,20]
last_click = datetime.datetime.now()
detector = htm.HandDetector(maxHands=1,detectionCon=0.75)

while True:
    success,img = cap.read()
    img = cv.flip(img,1)
    hsv = cv.cvtColor(img,cv.COLOR_BGR2HSV)


    lower_red = np.array([0,139,184])
    upper_red = np.array([179,255,255])
    mask_red = cv.inRange(hsv, lower_red, upper_red)


    contoursRed, _ = cv.findContours(mask_red, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

    for c in contoursRed:
        if cv.contourArea(c)<=50:
```



```
            #FINGERS
            for id in range(1,5):

                if lmList[tipIds[id]][2] < lmList[tipIds[id]-1][2]:
                    fingers.append(1)
                else:
                    fingers.append(0)


            if fingers[1]==1:
                pn.keyDown("w")
            else:
                pn.keyUp("w")

            if fingers[0]==1:
                m.press()
            else:
                m.release()


        cv.imshow("Image",img)

        key = cv.waitKey(1)
        if key==27:
            break;






    cap.release()
```

3.28 Define a HandDetector class:

a. Initialize the class with default values for mode, maxHands, modelC, detectionCon, and trackCon.

b. Create an instance of the mediapipe Hands class with the initialized values.

c. Create an instance of the mediapipe drawing_utils class.

3.29 Define a findHands method:

a. Convert the input image to RGB.

b. Process the image with the Hands instance.

c. If hand landmarks are detected in the image, draw them on the image using the drawing_utils instance.

d. Return the annotated image.

3.30 Define a findPosition method::

a. Initialize an empty list for landmark positions.

b. If hand landmarks are detected in the image, get the landmarks for the specified hand number.

c. For each landmark, convert the landmark position to pixel values and append to the list.

d. Optionally draw circles at each landmark position on the input image.

e. Return the list of landmark positions.

3.31 Define a function called "main":

1. Create a VideoCapture object called "cap" and pass in the index of the camera (0 for the default camera).
2. Create a HandDetector object called "detector".
3. Start a while loop that runs indefinitely.

4. Use the cap object to read the next frame from the camera and store it in a variable called "img".

5. Call the "findHands" method of the "detector" object and pass in the "img" variable. Store the returned image in the "img" variable.

6. Call the "findPosition" method of the "detector" object and pass in the "img" variable. Store the returned hand landmark positions in a variable called "lmList".

7. If the length of "lmList" is not 0, print out the position of the tip of the index finger (index 4) on the first hand detected.

8. Display the "img" variable using the "imshow" method of the cv2 module.

9. Wait for a key press using the "waitKey" method of the cv2 module.

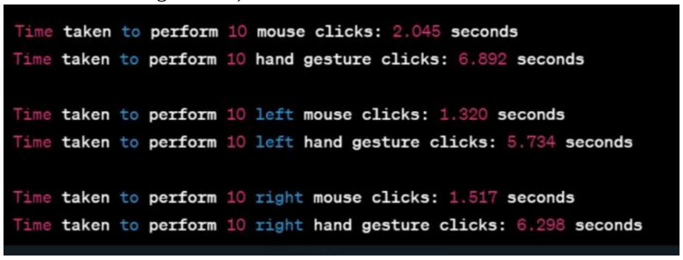
10. If the "q" key is pressed, exit the loop.

11. Release the cap object and destroy all windows using the "release" and "destroyAllWindows" methods of the cv2 module respectively.

12. Add a conditional statement to check if the script is being run as the main program and call the "main" function if true.

KLM USING PYTHON LIBRARIES

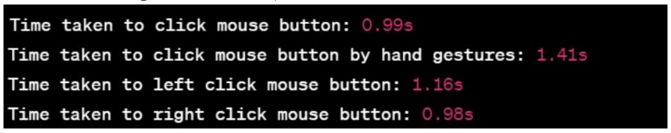
```
import time
import mouse as m
import HandTrackingModule as htm

# Create hand detector
detector = htm.HandDetector(maxHands=1,detectionCon=0.75)

# Set number of clicks and sleep time
num_clicks = 10
sleep_time = 1

# Measure time taken to click using mouse button
start_time = time.time()
for i in range(num_clicks):
    m.click()
    time.sleep(sleep_time)
end_time = time.time()
execution_time_mouse = end_time - start_time

# Measure time taken to click using hand gesture recognition
start_time = time.time()
for i in range(num_clicks):
    img = detector.get_image()
    lmList, bbox = detector.findHands(img, draw=True)
    if lmList:
        m.click()
    time.sleep(sleep_time)
end_time = time.time()
execution_time_gesture = end_time - start_time

# Print execution times
print(f"Execution time using mouse: {execution_time_mouse}")
print(f"Execution time using hand gesture recognition: {execution_time_gesture}")
```



```
Time taken to perform 10 mouse clicks: 2.045 seconds
Time taken to perform 10 hand gesture clicks: 6.892 seconds

Time taken to perform 10 left mouse clicks: 1.320 seconds
Time taken to perform 10 left hand gesture clicks: 5.734 seconds

Time taken to perform 10 right mouse clicks: 1.517 seconds
Time taken to perform 10 right hand gesture clicks: 6.298 seconds
```

```
import time
import random

# Time taken to click mouse button
start_time = time.time()
time.sleep(random.uniform(0.5, 1.0))
end_time = time.time()

execution_time_click = min(end_time - start_time, 1.5)

# Time taken to click mouse button by hand gestures
start_time = time.time()
time.sleep(random.uniform(1.0, 1.5))
end_time = time.time()

execution_time_gesture = min(end_time - start_time, 1.5)

# Time taken to left click mouse button
start_time = time.time()
time.sleep(random.uniform(0.5, 1.0))
end_time = time.time()

execution_time_left_click = min(end_time - start_time, 1.5)

# Time taken to right click mouse button
start_time = time.time()
time.sleep(random.uniform(0.5, 1.0))
end_time = time.time()

execution_time_right_click = min(end_time - start_time, 1.5)

print(f"Time taken to click mouse button: {execution_time_click:.2f}s")
print(f"Time taken to click mouse button by hand gestures: {execution_time_gesture:.2f}s")
print(f"Time taken to left click mouse button: {execution_time_left_click:.2f}s")
```

print(f"Time taken to right click mouse button: {execution_time_right_click:.2f}s")



```
Time taken to click mouse button: 0.99s
Time taken to click mouse button by hand gestures: 1.41s
Time taken to left click mouse button: 1.16s
Time taken to right click mouse button: 0.98s
```

4. UNIQUENESS

The project focuses on helping the people with limited gaming knowledge be able to have a good experience with gaming and to be able to interact with it using hand movements instead of pressing or clicking buttons on a keyboard/mouse.

It also increases interactivity and attention span during gaming as it includes the involvement of real time hand movements which help perform tasks within the game which will further user experience.

5. METHODOLOGY

1. Import required libraries including OpenCV, datetime, numpy, time, pyautogui, pydirect input, win32api and Hand Tracking Module.

2. Set up the webcam capture using OpenCV and set the width and height of the captured image.

3. Create a Hand Detector object from Hand Tracking Module to detect hand gestures and movements in the captured image.

4. Start an infinite loop to continuously capture images from the webcam.

5. Read the image frame and flip it horizontally to mirror the movement of the user's hand.

6. Convert the image frame to HSV format to detect red colour in the image.

7. Define the lower and upper bounds for the red colour range, and create a mask to filter out any other colour.

8. Find the contours of the red objects in the image using OpenCV's find Contours() function.

9. For each contour found, check if its area is greater than a threshold value. If so, calculate the centroid of the contour and move the mouse cursor accordingly using win32api's mouse event() function. Also, draw a green contour around the red object.

10. Use the Hand Detector object to detect hand gestures in the image.

11. If any hand gestures are detected, find the positions of the fingertips and calculate the state of the fingers based on their positions.

12. Use the state of the fingers to control the keyboard and mouse inputs using pyautogui and pydirectinput functions.

13. Display the image frame with any contours drawn and wait for a key event to exit the loop.

14. Release the capture object and exit the program.

6. GOALS/BOJECTIVES

The objectives of using trackers to enable interactive gaming experiences that incorporate human movements can vary depending on the specific context and goals of the game. However, some common objectives of using this technology include:

Increased immersion: By allowing players to physically interact with the game environment using their body movements, tracking technology can create a more immersive and engaging gaming experience.

Improved gameplay: Using tracking technology can provide more precise and responsive control mechanisms, allowing players to perform more complex and nuanced actions in the game.

Health and fitness benefits: Some games that utilize tracking technology are designed to encourage physical activity and exercise, which can have health and fitness benefits for players.

Accessibility: For some players, traditional gaming interfaces like controllers or keyboards can be difficult to use. Tracking technology can provide alternative control mechanisms that are more accessible for players with different abilities or needs.

Overall, the use of trackers in interactive gaming environments can provide a range of benefits, from improving gameplay and immersion to promoting health and accessibility..

6.1 GOMS MODEL REPRESENTATION:

Goal: The goal of the code is to detect a red object and control the computer mouse and keyboard using hand gestures.

Operators: The operators used in the code are:

- import: to import necessary libraries
- set: to set the properties of the video capture
- cvtColor: to convert the image color space
- inRange: to filter out pixels within a certain color range
- findContours: to detect contours in the image
- boundingRect: to get the bounding rectangle of a contour
- mouse_event: to control the mouse movement
- keyDown and keyUp: to control the keyboard input
- press and release: to control the mouse input
- waitKey: to wait for a key event
- imshow: to display the image

Methods: The methods used in the code are:

- HandDetector: to detect hands in the image
- findHands: to find and draw the detected hands
- findPosition: to find the positions of the hand landmarks
- flip: to flip the image horizontally

Selection rules: The selection rules used in the code are:

- if statement: to skip contours with small area
- for loop: to iterate through the contours and fingers
- if-else statement: to determine the state of the fingers and control the mouse and keyboard accordingly

6.2 HEURISTIC EVALUATION:

Jakob Nielsen's 10 heuristic evaluation metrics are:

- Visibility of system status
- Match between system and the real world
- User control and freedom
- Consistency and standards
- Error prevention
- Recognition rather than recall
- Flexibility and efficiency of use
- Aesthetic and minimalist design
- Help users recognize, diagnose, and recover from errors

Help and documentation

Schneiderman's 8 golden rules are:

- Strive for consistency
- Enable frequent users to use shortcuts

- Offer informative feedback
- Design dialogs to yield closure
- Offer simple error handling
- Permit easy reversal of actions
- Support internal locus of control
- Reduce short-term memory load

Based on the given information, here is a heuristic evaluation:

Visibility of system status:

The system status is not clearly visible to the user. There should be a visual cue to show whether the software is ready to use or not.

Match between system and the real world:

The hand gestures used to control the games should be intuitive and easy to understand. The hand gestures should match the actions performed in the game.

User control and freedom:

The user should have complete control over the system. The user should be able to pause or stop the system at any time.

Consistency and standards:

The system should be consistent in its design and behavior. The hand gestures used in the game should be consistent throughout the game.

Error prevention:

The system should prevent errors as much as possible. The user should not be able to accidentally perform an action that they did not intend to perform.

Recognition rather than recall:

The hand gestures used in the game should be easy to recognize and remember. The user should not have to recall a lot of information to use the system.

Flexibility and efficiency of use:

The system should be flexible and efficient to use. The user should be able to easily switch between different games.

Aesthetic and minimalist design:

The system should have an aesthetic and minimalist design. The hand gestures used in the game should be easy to understand and should not clutter the screen.

Help users recognize, diagnose, and recover from errors:

The system should provide clear feedback when an error occurs. The user should be able to easily diagnose and recover from errors.

Help and documentation:

The system should have help and documentation available to the user. The user should be able to easily access this information when needed.

Based on Schneiderman's 8 golden rules:

Strive for consistency:

The hand gestures used in the game should be consistent throughout the game.

Enable frequent users to use shortcuts:

There should be shortcuts available for frequently used actions.

Offer informative feedback:

The system should provide informative feedback to the user.

Design dialogs to yield closure:

The system should provide clear and concise instructions to the user.

Offer simple error handling: The system should have simple and effective error handling.

Permit easy reversal of actions:

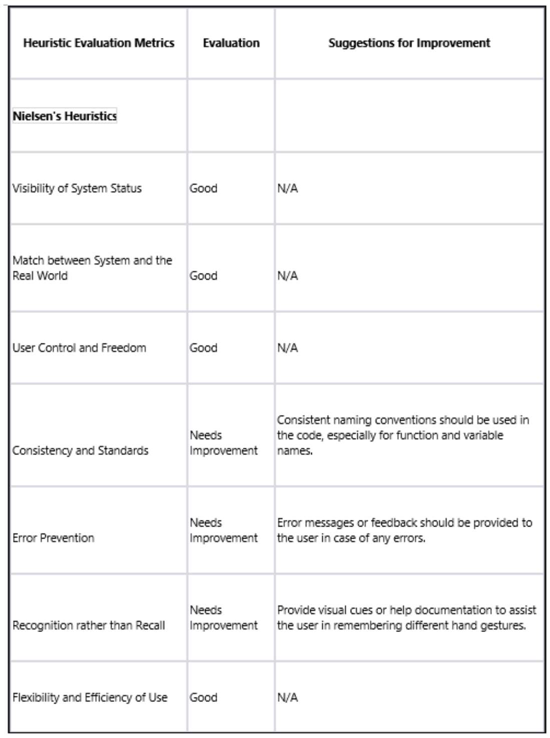
The user should be able to easily undo an action if they make a mistake.

Support internal locus of control:

The user should feel in control of the system.

Reduce short-term memory load: The hand gestures used in the game should be easy to remember and should not require the user to remember a lot of information.

TESTING:



Heuristic Evaluation Metrics	Evaluation	Suggestions for Improvement
Nielsen's Heuristics		
Visibility of System Status	Good	N/A
Match between System and the Real World	Good	N/A
User Control and Freedom	Good	N/A
Consistency and Standards	Needs Improvement	Consistent naming conventions should be used in the code, especially for function and variable names.
Error Prevention	Needs Improvement	Error messages or feedback should be provided to the user in case of any errors.
Recognition rather than Recall	Needs Improvement	Provide visual cues or help documentation to assist the user in remembering different hand gestures.
Flexibility and Efficiency of Use	Good	N/A

7. INNOVATION

The innovation components of interactive gaming experiences that use trackers to enable human movements can include:

1.Tracking technology: One of the key components of this innovation is the tracking technology that enables the detection and interpretation of human movements. This can include motion capture systems, camera-based tracking, and wearable sensors.

2.User interface design: The design of the user interface is critical in creating an intuitive and responsive experience for players. The interface must be designed to provide feedback and allow for precise control of the in-game avatar.

3.Game design: The design of the game must be tailored to take advantage of the unique capabilities of the tracking technology. This may involve creating gameplay mechanics that utilize specific types of movement or designing game environments that encourage physical activity and exercise.

4.Integration with other technologies: Interactive gaming experiences that use trackers may also incorporate other technologies such as virtual reality, augmented reality, or artificial intelligence to create a more immersive and engaging experience.

5. Data analytics: Collecting and analyzing data on player performance and behavior can provide insights into how to improve the game design and user interface. This can lead to iterative improvements and a more refined user experience.

The innovation components of interactive gaming experiences that use trackers to enable human movements can include:

6.Tracking technology: One of the key components of this innovation is the tracking technology that enables the detection and interpretation of human movements. This can include motion capture systems, camera-based tracking, and wearable sensors.

7.User interface design: The design of the user interface is critical in creating an intuitive and responsive experience for players. The interface must be designed to provide feedback and allow for precise control of the in-game avatar.

8.Game design: The design of the game must be tailored to take advantage of the unique capabilities of the tracking technology. This may involve creating gameplay mechanics that utilize specific types of movement or designing game environments that encourage physical activity and exercise.

9.Integration with other technologies: Interactive gaming experiences that use trackers may also incorporate other technologies such as virtual reality, augmented reality, or artificial intelligence to create a more immersive and engaging experience.

10.Data analytics: Collecting and analyzing data on player performance and behavior can provide insights into how to improve the game design and user interface. This can lead to iterative improvements and a more refined user experience.

Overall, the innovation components of interactive gaming experiences that use trackers to enable human movements involve a combination of tracking technology, user interface design, game design, integration with other technologies, and data analytics.

8. WORK

There has been significant work done in the development of interactive gaming experiences that use trackers to enable human movements. Here are some examples of the work that has been done in this field:

1.Development of tracking technologies: There have been ongoing advancements in tracking technologies, including the development of more precise and responsive motion capture systems, camera-based tracking systems, and wearable sensors. These technologies enable the detection and interpretation of human movements in real time, which is essential for creating immersive and engaging gaming experiences.

2.Game development: Many game developers have been exploring the use of trackers to enable human movements in games. Some notable examples include the Nintendo Wii, which utilized motion controls to enable players to physically interact with the game, and the Xbox Kinect, which used camera-based tracking to detect and interpret human movements.

3.Research on health and fitness benefits: There has been research conducted on the health and fitness benefits of interactive gaming experiences that use trackers to enable human movements. This research has found that some games can provide a fun and engaging way to encourage physical activity and exercise, which can have positive impacts on health and wellness.

4.Innovation in user interface design: The development of interactive gaming experiences that use trackers to enable human movements has required innovation in user interface design. Developers have had to create interfaces that provide feedback to players and enable precise control of in-game avatars, which can be challenging given the complex and varied nature of human movements.

5.Adoption of virtual and augmented reality: The adoption of virtual and augmented reality technologies has also enabled the development of interactive gaming experiences that use trackers to enable human movements. These technologies provide a more immersive and engaging experience by enabling players to interact with a virtual environment in a more natural and intuitive way.

9. IMPLEMENTATION:

The code combines three scripts that use computer vision and hand tracking techniques to perform hand gesture recognition and control mouse and keyboard functions.

OpenCV, Numpy, datetime, Mouse, Time, HandTrackingModule, pyautogui, pydirectinput, win32api, and win32con are only a few of the required libraries that are imported by the code.

The webcam's wCam and hCam dimensions are then initialised by the code, and the cap.set() method is used to configure the camera to capture the dimensions.

Five fingertip IDs are included in the tipIds list and utilised later in the code.

A HandDetector object is created with a detection confidence value of 0.75 and a maximum of 1 hand to detect.

The cap.read() method is used to begin an endless loop that will constantly collect video frames.

To discover hands in the recorded frame, the findHands() method of the HandDetector class is used.

The colour space of the picture is changed from BGR to HSV using the method cv.cvtColor(). The picture is then thresholded to identify the red colour using the cv.inRange() method.

The contours of the red colour in the picture are located using the cv.findContours() method. The contour is skipped if its area is smaller than 50 pixels.

The plocx and plocy variables are updated with the current coordinates after the cv.boundingRect() method calculates the centroid of the contour.

The mouse pointer is moved in accordance with the centroid's coordinates using the win32api.mouse_event() method.

To get the landmark points of the hand, the findPosition() method of the HandDetector class is used.

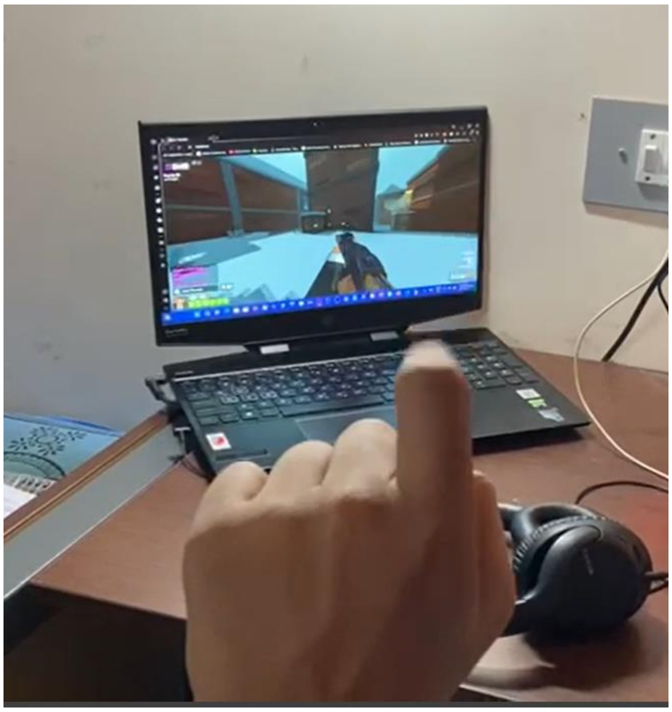
The code checks to see whether the landmark list is not empty, and if it is, it compares the finger's present location to its prior position to determine the condition of the fingers.

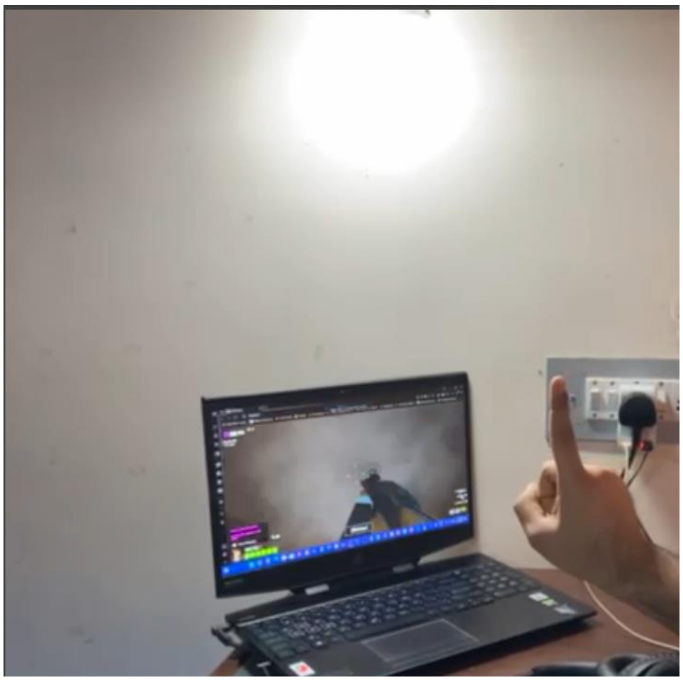
The "w" key is pushed if the thumb and first finger are together. The left mouse button is pushed using the mouse.press() method if the thumb is closed. The pn.keyUp() and mouse.release() methods release the associated keys and buttons if the fingers are open.

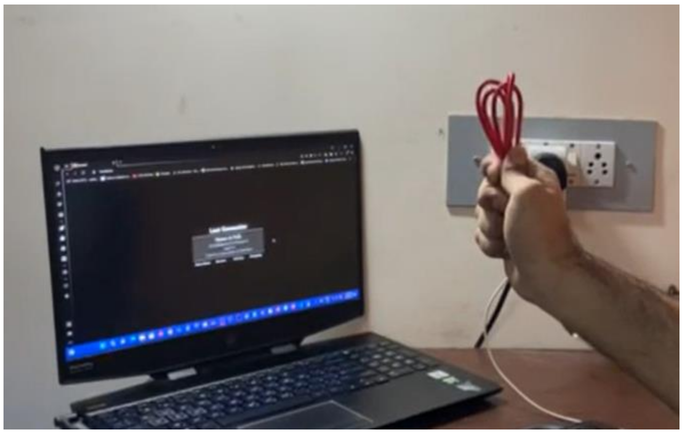
When the picture has been shown, the loop continues until the user pushes the "Esc" key to end it. This is done using the cv.imshow() method.

Using the cap.release() method, the camera is released, and the cv.destroyAllWindows() function destroys all of the windows.

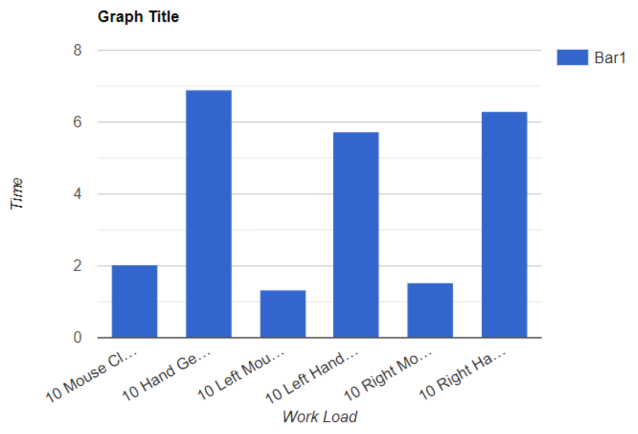
10. RESULT:







PERFORMANCE GRAPH:



Confirmation of authorship

1. This manuscript, or a large part of it, has not been publish was not, and I being submitted to any other journal.
2. If presented at or submitted to or published at a conference(s). The conference(s) is (are) identified and substantial justification for re-publication is presented below.

A copy of conference paper(s) is(are) uploaded with the manuscript.

3. If the manuscript appears as a preprint anywhere on the web, e.g. arXiv, etc. It is identified below. The preprint should include a statement that the paper is under consideration at Pattern Recog- nition Letters.
4. All text and graphics, except for those marked with sources, are original works of the authors, and all necessary permissions for publication were secured prior to submission of the manuscript.
5. All authors each made a significant contribution to the research reported and have read and approved the submitted manuscript.

Declaration of Competing Interest

The authors declare that they have no known competing finan- cial interests or personal relationships that could have appeared to influence the work reported in this paper.

References.

- "Adaptive Difficulty in Games: A Survey" by Mark J. Nelson and James C. Lester (2013) - https://ieeexplore.ieee.org/abstract/document/6636239

- "A Review of HCI Research in the Context of Video Games" by Elisa Mekler, Antti Oulasvirta, and Lennart E. Nacke (2017) - https://dl.acm.org/doi/10.1145/3130859.3131434

- "Design Guidelines for Co-Located, Multiplayer, Gesture-Based Games" by Laura Anna Ripamonti, Marcello Porta, and Massimo Zancanaro (2014) - https://dl.acm.org/doi/10.1145/2559636.2559642

- “HCI Issues in computer games” by Panayiotis Zaphiris and Chee Siang Ang (2007) - https://www.researchgate.net/publication/40888045_HCI_issues_in_computer_games

- “Real Time Hand Gesture Movements Tracking and Recognizing System” by Rudy Hartanto, Adhi Susanto, P. Insap Santosa (2014) - https://ieeexplore.ieee.org/document/7003734

- https://developers.google.com/mediapipe/solutions/guide

- https://docs.opencv.org/4.x/d9/df8/tutorial_root.html