Diploma in

Mechatronics Engineering

Final Year Project Report

Project Title:

Development of the Control System for an Outreach Robot

Project Supervisor:

Dr Edwin Foo

Project Student:

Takesawa Saori (170319P)

Abstract

In the recent years, the demand and use of robots is increasing rapidly. Hence, future engineers need to be cultivated and one of the ways is to have children develop interest in robotics through interacting with robots. To do this, the software of robot that can play Scissor Paper Stone game with the user is developed. The software interacts with the user through Ubuntu OS, GUI and video streams from connected webcam. It crops and processes the frame of the video stream to a binary image with skin extraction to get the move of the user when the software indicates to put a move to the user. It feeds the binary image to the trained Convolutional Neural Network model to predict the move and gives the result of the game by comparing with its own randomly chosen gesture. This software requires user to put their gesture's front view clearly, lacking flexibility. More can be done to improve this software.

Acknowledgements

I would like to take this opportunity to express my appreciation to my project supervisor, Dr Edwin Foo, who patiently guided and gave useful suggestion throughout this whole project. Without his help, it would be very difficult to complete this project.

Furthermore, I would also like to express my appreciation to Dr Benjamin Ma, who gave permission to use certain equipment and gave advice for the project.

Lastly, I would like to acknowledge with much appreciation to lecturers and my friends who gave me advice on how to improve the Scissor Paper Stone game.

Table of Contents

Introduction

In recent years, the demand for robots (like robot arm) in the industries is rapidly increasing due to the many advantages of robots.

Advantages include:

- High productivity with long operation hours
- Resolve shortage of skill labour issue
- Flexible operations
- Consistent quality in production
- Able to work in hazardous environment to human
- And more…

As the demand increases, the need for engineers to develop and troubleshoot robots will also increase. Hence, to cope with the demand, we need more children to develop interest in robotics and take up studies and jobs of robotics.

To develop interest in children, interacting with robots, like playing games with them, will be effective.

Objective

The project aims to develop an engaging and interactive software that can play scissor paper stone with the user. The software uses video stream to get the gesture of the user and responds to the user with a gesture.

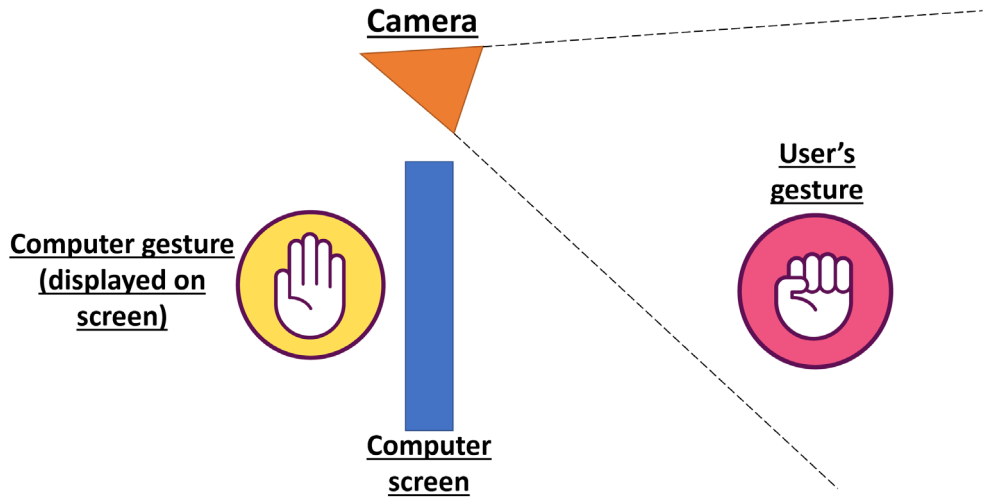


Figure 1 Simple diagram to illustrate how the software run

The software needs to be able to:

- Recognise the gesture of the user (Scissor, Paper, Stone) through a webcam with more than 90% accuracy.
- Tell the user the gesture that the software has predicted.
- Respond to the gesture of the user with another gesture from the software within 1 second.
- Have a Graphics User Interface, GUI, of the game that is engaging, easy to understand and use for the user.

Project Scope

Key Specification

The software uses the following to run:

- Software
  - Ubuntu 18.04
  - Python 3.7.3
  - Tensorflow 1.13.1
  - Keras 2.2.4
  - OpenCV 4.0.0
  - Pygame 1.9.6
- Hardware
  - Logitech HD Pro Webcam C910

Software

Operating System

Ubuntu 18.04 is the operating system used to run the software. The software can also be run on Windows 10 however, initializing the webcam takes some time. This makes the software hang momentarily hence, running the software on Ubuntu is preferred.

Figure 2 Ubuntu Logo

Programming Language

Python is used to write all the programs for the software. Python is an object-oriented, general purpose and high-level programming language. It has a simple syntax that allow easier troubleshooting and less typing, hence it was used.

Figure 3 Python Logo

Software Libraries

Deep Learning

- **Tensorflow**
  - Tensorflow is a free and open source library for developing and training Machine Learning (ML) model. It is used to train the Convolutional Neural Network model to recognise the user gesture.

Figure 4 Tensorflow Logo

- **Keras**
  - Keras is an open source high-level neural network API written in Python. It is capable of running on top of Tensorflow, hence used to train the Convolutional Neural Network model (refer to Concepts) to recognise the user gesture using Tensorflow backend. Using Tensorflow directly is possible but was not done as writing the program using Keras is much easier.

Figure 5 Keras Logo

Computer Vision

- **OpenCV**
  - OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. It is used to get the video stream from the webcam, draw figures into the video and process the video frames to feed them to the CNN model.

Figure 6 OpenCV Logo

Graphics User Interface (GUI)

- **Pygame**
  - Pygame is a cross-platform set of Python modules for writing video games. It includes Python libraries for computer graphics and sound. It is used to create the GUI for the Scissor Paper Stone game.

Figure 7 Pygame Logo

Hardware

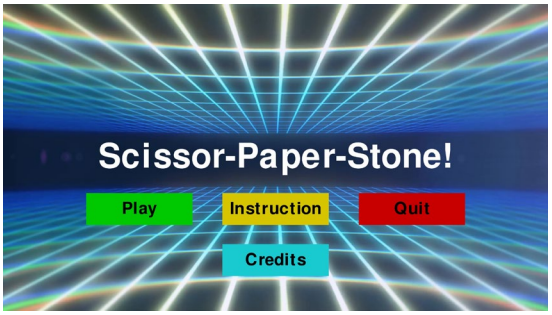
Logitech HD Pro Webcam C910 (1080p) is used to get the video stream for the GUI, processing of image and getting the frame with the gesture of the user.

Figure 8 Logitech HD Pro Webcam C910

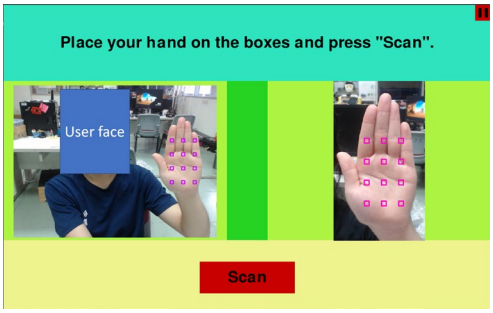
Major Deliverables

The software will run the game in this manner:

The software will open game Main Menu. The 'Play' button is pressed.

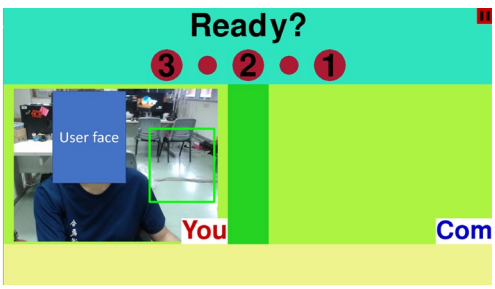


The software will open Scan screen. There will be instruction on where the user's hand is to be placed and the 'Scan' button is pressed when user has placed their hand.

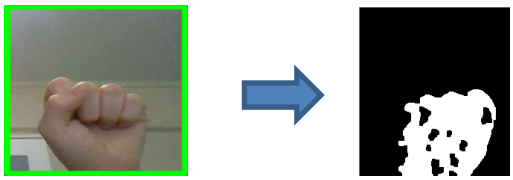


When the 'Scan' button is pressed, the last video frame before the button is pressed is converted from RGB to HSV. Hue and saturation histogram of the skin is calculated using the colour in the pink rectangles.

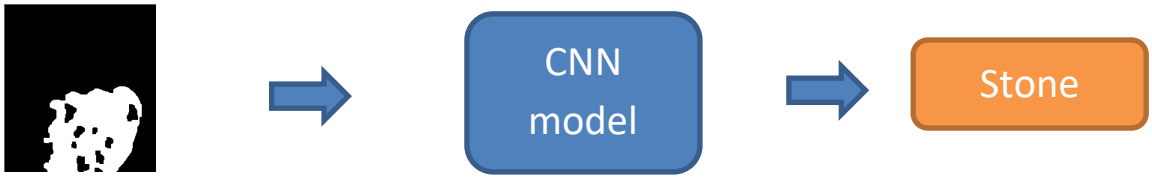
The software will open Game screen. There will be words and circles to help user gauge the timing to put their move.



When the user is expected to put a move in the green box, the software crops the frame at the green box, process it into binary image with only the skin present as white.

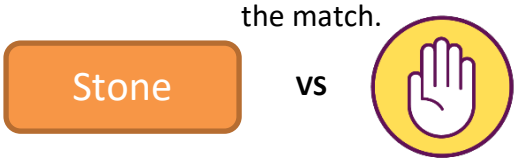


The binary image is fed to the CNN model to predict the gesture.



The software chooses a move (Scissor, Paper, Stone) randomly.

The software compares the move between the software and the user to see who won the match.



The software displays the results and the moves for both sides with the frame that was used for gesture prediction.

Gantt Chart

No.	Activity		Wk 1	Wk 2	Wk 3	Wk 4	Wk 5	Wk 6	Wk 7	Wk 8	Wk 9	Wk 10	Wk 11	Wk 12
1	Set up • Setup necessary softwares • Setup virtual environment	Actual												
		Plan												
2	Knowledge Acquisition • Python programming • Keras • Basics of Convolution Neural Network (CNN) • Programming using 3D camera	Actual												
		Plan												
3	Collecting and Organising Data • Download image datasets on the Internet • Take pictures of necessary hand gestures • Organising image data into their respective folder	Actual												
		Plan												
4	Implementation & Learning • Write simple deep learning programs • Learning the function and problems occured • Troubleshooting problems in the programs	Actual												
		Plan												
5	Experimenting & fine-tune program • Run the deep learning program several times with different parameters (no. of layers, optimizer, etc.) and pretrained models • Fine-tune program	Actual												
		Plan												
6	Learning & Comparison • See if objection detection YOLO can be used • See if Tensorflow objection detection API can be used	Actual												
		Plan												
7	Skin segmentation detection & GUI • Learn and apply skin segmentation • Detection for gesture using binary images from segmentation • Create GUI for the game using Pygame	Actual												
		Plan												

Project Implementation

Software Development

Concepts

To recognise user's gesture, I have tried different method. Below are methods that I tried using:

1. Trained CNN model from scratch with grayscale images.

   Reasons:

   a. Colour images' pixel values are greatly affected when there is a change in lighting. This may cause difficulty for the model to recognize gesture as the model may not be used to certain lighting. For example, the model can recognize gestures under white lighting but not orange lighting. To avoid this, the model needs to be trained on images with various lighting. However, there will be infinite variation of lighting and gathering such infinite number of images is not realistic. On the other hand, grayscale images only contain information of the intensity of the lighting. This means the model will not be influenced by various colours of lighting and the amount of training images needed will be greatly reduced. This makes the training much easier.

   b. Colour images have more than 1 colour channel, multiplying the data to be processed during training and application. This would make both training the model and recognition slower. Hence, grayscale images are used.

   Outcome:

   - Accuracy of the model is no more than 70%.

   Due to the reason above, this method was not used.

2. Use binary images that contains only the edges of the image to be used for the CNN model training. These images are created using Canny Edge Detection.

   Reason:

   a. The input image is greatly simplified, making the model easier for it to predict gesture.
   b. There is much lesser data to process compared to coloured images since the image is binary image that only has one channel.

Outcome:

- The edges of many images are not created as they had blur areas. But these images are not blur to the extent that is not recognizable.
  This is an example of such image:

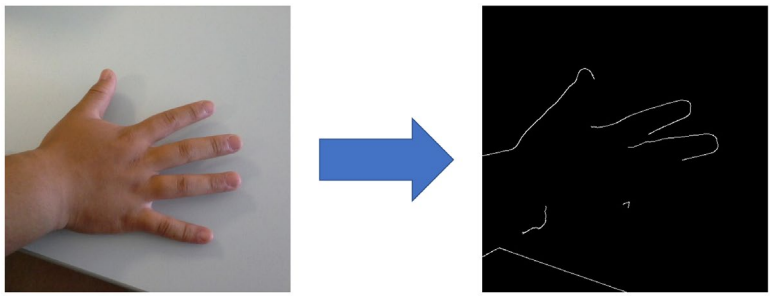


Figure 9 Example of Canny Edge Detection limitation

There would be cases where blurry image is captured to recognise gesture. With this method, recognition would not be stable and accurate. This method lack flexibility and hence, this method was not used.

3. Trained Haar Cascade with grayscale images of hand.

Reason:

a. Haar Cascade will give coordinate for the box that is bounding the hand. I thought of using that coordinate to crop out the hand from the image and feed that to the CNN model. By doing this, only necessary data is fed, and the remaining data would not 'confuse' or decrease the accuracy of the CNN model.

Outcome:

- Haar Cascade produced many false bounding boxes even when there is no hand present. This is probably due to the complex of a hand. As a hand can have many different poses, it would be hard for the Haar Cascade to learn what is a hand. To do this, a huge number of images is needed to get a very accurate cascade.

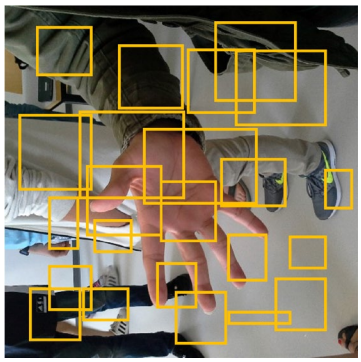


Figure 10 Haar Cascade training result

Since doing this method was not reasonable to continue, this method was not used.

4. Trained CNN model with transfer learning with RGB images.

   Reason:

   a. With transfer learning, it will greatly speed up the training and obtain higher accuracy more easily.
   b. RGB images were used as there are no pre-trained model that is trained with grayscale images.

   Outcome:

   - The model has correct prediction with high confidence (more than 90%) for paper and stone gesture. However, at a certain angle or/and with scissor gesture, the model made wrong prediction with high confidence as well.
   - The model is predicting gestures with high confidence when there is nothing in the image. This is because the model is not given any other option other than scissor, paper and stone. The model is forced to 'choose' out of these three options when predicting. The reason for the high confidence is not known, but it is probably also due to the limited option.

   Due to the above reasons, this method is not used.

5. Trained CNN model with transfer learning with RGB images. An unknown class is added to the class by feeding in random images (buildings, clothes, etc.) without any hand in the image for the unknown class.

   Reason:

   a. I thought that by introducing the unknown class, the CNN model would be able to learn the presence of the hand in the image. If this can be done, it would greatly increase the accuracy of the model.
   b. I hoped to solve the problem raised in method 4, where the model predicts a gesture when there is no gesture in the image.

   Outcome:

   - Compared to method 4, the mistakes with the prediction has gone worst. The model now predicts most gestures as the unknown class.

   Due to the above reason, this method is not used.

6. Trained Tensorflow Object Detection API to detect gesture using manually tagged images of gestures.

   Reason:

   a. Using the object detection, the detection of gesture would be very robust and it will not be forced to predict any gesture when there is no gesture present.

Outcome:

- The object detection API has successfully detected gesture very accurately with bounding box like below.

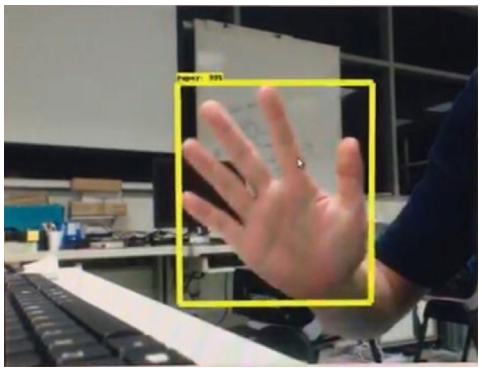


Figure 11 Tensorflow Object Detection API result

However, it takes about 2 seconds to predict one frame, even with a GTX 1070 (8 GB). As it does not meet the objective, this method is not used.

7. Trained CNN model from scratch with skin segmented binary images. The skin segmentation is done by back projection. The frame is converted to HSV then skin colour histogram is taken to do back projection.

   Reason:

   a. I can decide when the model should be used to predict using the percentage of white. Skin segmented binary image colours the skin colour white. By setting the percentage of white to finally enable to the model to predict, I can solve the problem with the model predicting gestures when no gesture is present.
   b. The input image is greatly simplified, making the model easier for it to predict gesture.
   c. Skin segmentation would be robust as Value of HSV (also known as luminosity) is not used for the back projection. Doing this will enable the skin segmentation to be flexible with slight changes with lighting.

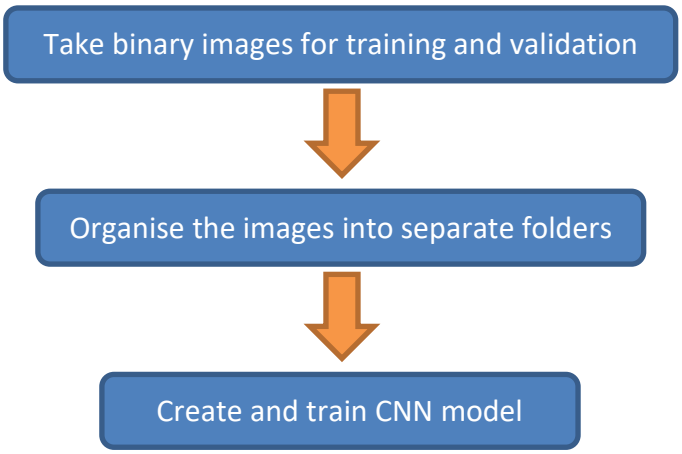
   Outcome:

   - The model takes at most 10 milliseconds to predict accurately with more than 95% confidence.
   - The model is not used when no gesture is inside.

This method has been used since it has no major problems.

Training CNN model

The following steps were taken to train the CNN model:



Take binary images for training and validation

To take binary images, I have created a program to do this.

Full Main code:

```
import os  # importing os module
import cv2  # importing OpenCV

init = True
run = True
cap = cv2.VideoCapture(0)
while run:
    ret, frame = cap.read()
    cv2.normalize(frame, frame, 0, 255, cv2.NORM_MINMAX)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        cv2.destroyAllWindows()
        run = False
    if ret:
        if init:
            frame = translucent_obj(frame, draw_rect_right, 1.0)
            cv2.imshow('frame', frame)
            if cv2.waitKey(1) & 0xFF == ord('a'):
                start_rect, end_rect = draw_rect_right(frame, True)
                hand_hist = set_hand_hist(frame, start_rect, end_rect)
                init = False
                cv2.destroyWindow('frame')
        else:
            crop = frame [125:355, 425:655]
            cv2.rectangle(frame, (425, 125), (655, 355), (0, 255, 0), 5)
```

```
            frame = translucent_obj(frame, draw_rect_right, 1.0)
            cv2.imshow('frame', frame)
            if cv2.waitKey(1) & 0xFF == ord('a'):
                start_rect, end_rect = draw_rect_right(frame, True)
                hand_hist = set_hand_hist(frame, start_rect, end_rect)
                init = False
                cv2.destroyWindow('frame')
        else:
            crop = frame [125:355, 425:655]
            cv2.rectangle(frame, (425, 125), (655, 355), (0, 255, 0), 5)
            thresh = hist_mask(crop, hand_hist)
            clear = clearer(thresh, 5, 1)
            cv2.imshow('clear', clear)
            cv2.imshow('crop', crop)
            cv2.imshow('frame', frame)
            if cv2.waitKey(1) & 0xFF == ord('r'):
                  cv2.imwrite('rock/rock'+str(count_rock)+'.jpg', clear)
                  count_rock = count_rock + 1
            if cv2.waitKey(1) & 0xFF == ord('p'):
                  cv2.imwrite('paper/paper'+str(count_paper)+'.jpg', clear)
                  count_paper = count_paper + 1
            if cv2.waitKey(1) & 0xFF == ord('s'):
                   cv2.imwrite('scissor/scissor'+str(count_scissor)+'.jpg', clear)
                   count_scissor = count_scissor + 1
cv2.destroyAllWindows()
cap.release()
```

First, I imported the **os** module to be able to interface the underlying operating system that the program is running on. (The program is run on Ubuntu). I also imported OpenCV.

```
import os  # importing os module
import cv2  # importing OpenCV
```

Set constants to control loops in the program.

```
init = True
run = True
```

Preparing the connected webcam to start taking frames.

```
cap = cv2.VideoCapture(0)
```

Go into infinite loop.

```
while run:
```

Put the frame from the video stream and the availability of frame into frame and ret respectively. (ret will return False if no frames is obtained)

```
    ret, frame = cap.read()
```

Process the frame to make it clear by normalizing it.

```
cv2.normalize(frame, frame, 0, 255, cv2.NORM_MINMAX)
```

Set the program to close if ‘q’ key is pressed.

```
if cv2.waitKey(1) & 0xFF == ord('q'):
    cv2.destroyAllWindows()
    run = False
```

Check if frame from the webcam is taken.

```
if ret:
```

Check if the initialization (taking of skin histogram) needs to be done:

```
    if init:
        # True if skin histogram is not taken yet
```

Draw squares to indicate where to place hand to take skin histogram.

```
        frame = translucent_obj(frame, draw_rect_right, 1.0)
        # Function
```

Display the video stream with the squares.

```
        cv2.imshow('frame', frame)
```

If ‘a’ key is pressed, take the colour (skin) histogram bounded by the squares.

```
        if cv2.waitKey(1) & 0xFF == ord('a'):
            start_rect, end_rect = draw_rect_right(frame, True)
            # Function to get the coordinates of the squares
            hand_hist = set_hand_hist(frame, start_rect, end_rect)
            # Function to get colour histogram
            init = False
            # set initialization to False to indicate initialization is done
            cv2.destroyWindow('frame')
            # close the window created to show the video stream
```

If not initialization, crop the right side of the frame. Cropping is done so that other objects (having similar colour to skin) like the face of the user does not interfere when predicting with CNN model.

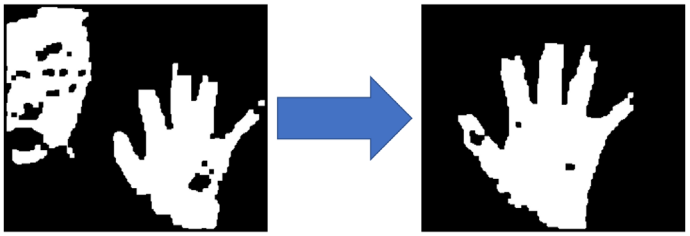


Figure 12 Binary images without and with cropping

```
    else:
        cv2.rectangle(frame, (425, 125), (655, 355), (0, 255, 0), 5)
        # draw rectangle to indicate the cropped area and where to place hand
        crop = frame [125:355, 425:655]
        # crop the right side of the frame
```

Back project the skin.

```
        thresh = hist_mask(crop, hand_hist)
        # Function to get the binary skin segmented image
```

Process to the binary image to reduce noise.

```
        clear = clearer(thresh, 5, 1)
        # Function to get a clearer binary image
```

Display the video stream, the cropped area of frame and the binary skin segmented image.

```
        cv2.imshow('clear', clear)
        cv2.imshow('crop', crop)
        cv2.imshow('frame', frame)
```

Save the binary image to the rock, paper, scissor folder when pressed ‘r’, ‘p’ or ‘s’ key respectively. Also increment count for numbering the images’ name.

```
        if cv2.waitKey(1) & 0xFF == ord('r'):
            cv2.imwrite('rock/rock'+str(count_rock)+'.jpg', clear)
            count_rock = count_rock + 1
        if cv2.waitKey(1) & 0xFF == ord('p'):
            cv2.imwrite('paper/paper'+str(count_paper)+'.jpg', clear)
            count_paper = count_paper + 1
        if cv2.waitKey(1) & 0xFF == ord('s'):
             cv2.imwrite('scissor/scissor'+str(count_scissor)+'.jpg', clear)
             count_scissor = count_scissor + 1
```

Close all windows created during the program.

```
cv2.destroyAllWindows()
```

Release the webcam from capturing frames.

```
cap.release()
```

Organise the images into separate folders

The binary images are organised as the following in the images folder:

images

- train
  - c0 (994 'Stone' images)
    - rock1.jpg
  - c1 (999 'Paper' images)
    - paper1.jpg
  - c2 (998 'Scissor' images)
    - scissor1.jpg
- test
  - c0 (197 'Stone' images)
    - rock1001.jpg
  - c1 (199 'Paper' images)
    - paper1001.jpg
  - c2 (198 'Scissor' images)
    - scissor1001.jpg

Create and train CNN model

CNN training program's full code:

```
import tensorflow as tf
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)

import os
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import EarlyStopping
from PIL import Image

train_data_dir = r'C:\Users\8051\Desktop\Saori\hand dataset\binary\images\train'
validation_data_dir = r'C:\Users\8051\Desktop\Saori\hand dataset\binary\images\test'
model_save_path = r'C:\Users\8051\Desktop\Saori\hand dataset\binary\binary_CNN_model.h5'
weight_save_path = r'C:\Users\8051\Desktop\Saori\hand dataset\binary\binary_weight.h5'

img_width, img_height = 100, 100
epochs = 50
batch_size_train = 128
batch_size_test = 32
nb_train_samples = 2981
nb_validation_samples = 594
```

```
model = Sequential()
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(100, 100, 1)))
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D((2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
```

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size_train,
    class_mode='categorical',
    color_mode='grayscale')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size_test,
    class_mode='categorical',
    color_mode='grayscale')
```

```
model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size_train,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size_test)

model.save_weights(weight_save_path)
model.save(model_save_path)
```

The first few lines of codes are only for Windows OS. Without these lines in Windows, Tensorflow cannot be used.

```
import tensorflow as tf
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)
```

Importing necessary modules and functions from libraries.

```
import os
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import EarlyStopping
from PIL import Image
```

Stating paths to the training and testing image folders. Also setting path for the model and the weights.

```
train_data_dir = r'C:\Users\8051\Desktop\Saori\hand dataset\binary\images\train'
validation_data_dir = r'C:\Users\8051\Desktop\Saori\hand dataset\binary\images\test'
model_save_path = r'C:\Users\8051\Desktop\Saori\hand dataset\binary\binary_CNN_model.h5'
weight_save_path = r'C:\Users\8051\Desktop\Saori\hand dataset\binary\binary_weight.h5'
```

Setting image dimension, number of epochs, batch size for training and validation, and lastly, stating the number of training and testing images.

```
img_width, img_height = 100, 100
epochs = 50
batch_size_train = 128
batch_size_test = 32
nb_train_samples = 2981
nb_validation_samples = 594
```

Creating CNN model network architecture.

```
model = Sequential()
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(100, 100, 1)))
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D((2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
```

Compile with categorical cross entropy loss and adam optimizer.

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Augmenting training images. Rescaling is also done so that the model deals with small numbers instead of large numbers.

```
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)
```

Rescaling testing images.

```
test_datagen = ImageDataGenerator(rescale=1. / 255)
```

Generating training and validation data.

```
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size_train,
    class_mode='categorical',
    color_mode='grayscale')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size_test,
    class_mode='categorical',
    color_mode='grayscale')
```

Fit the model.

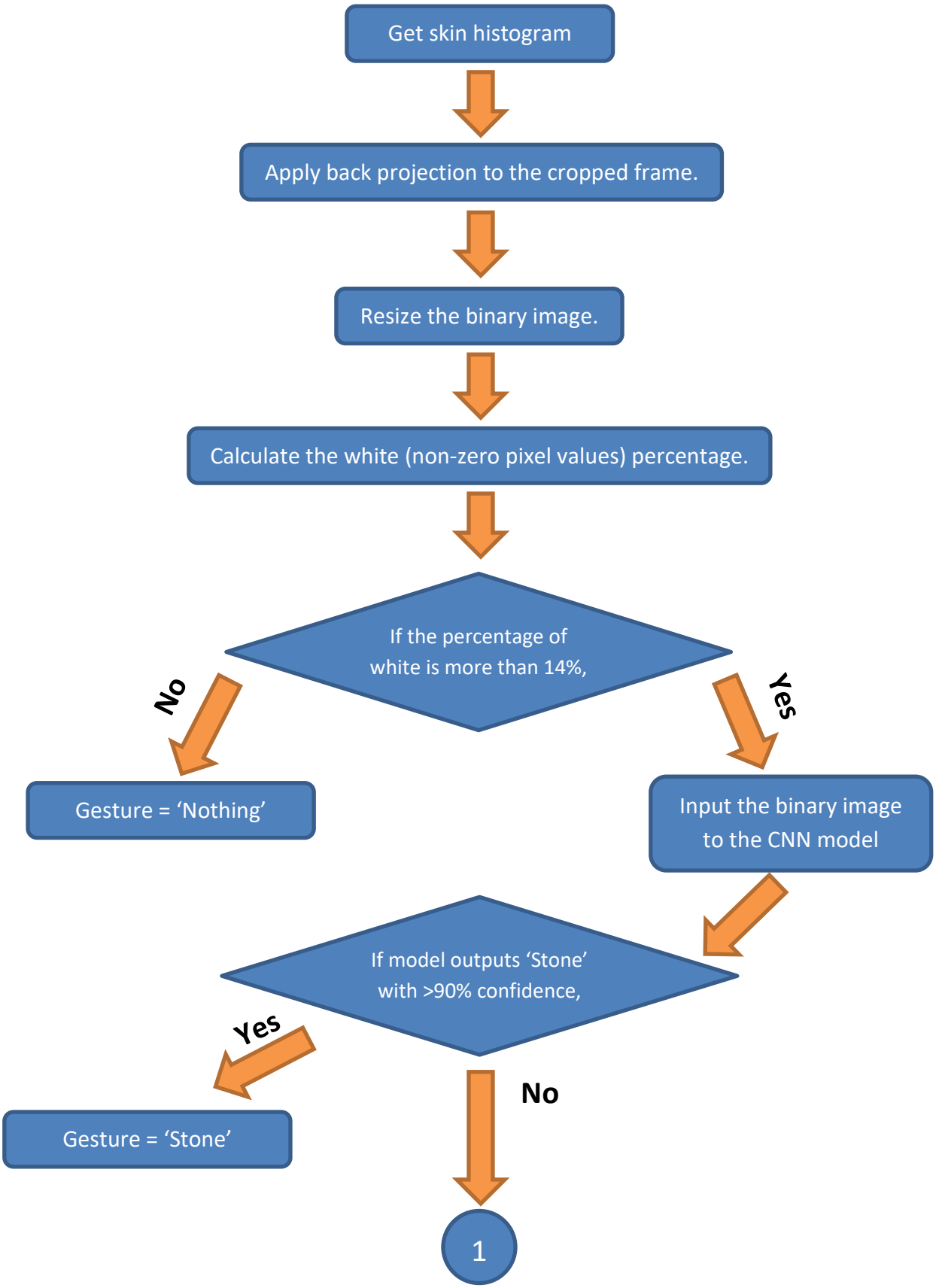
```
model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size_train,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size_test)
```

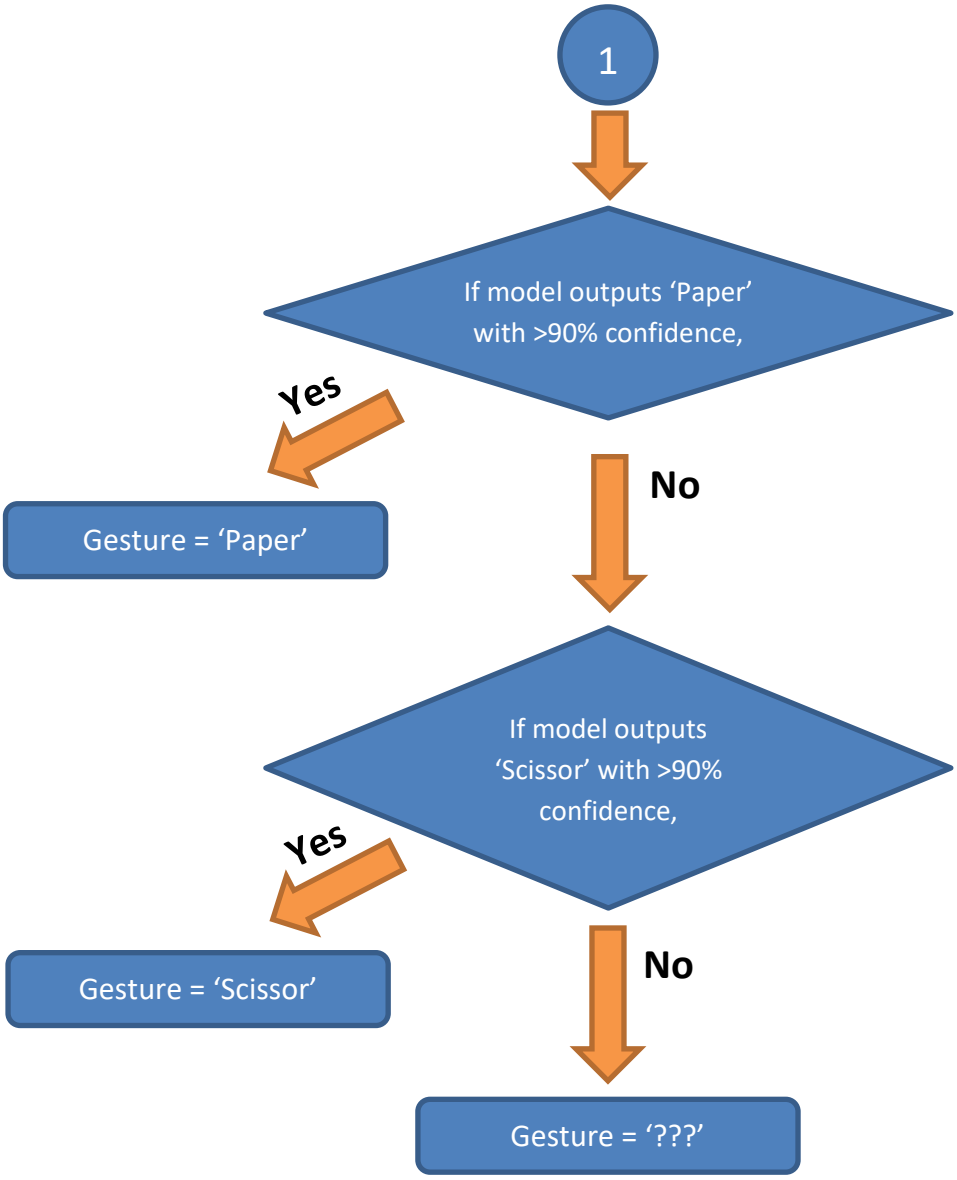
Save the model and the weight to their respective path.

```
model.save_weights(weight_save_path)
model.save(model_save_path)
```

Prediction from CNN model

To get the prediction from the CNN model, these are the steps to take:





Code:

```
import tensorflow as tf
import cv2
import numpy as np

model = tf.keras.models.load_model('CNN/binary_CNN_model.h5')

clear = crop_binary(frame)
# Function to get a cropped binary clear image
white_ratio, h_clear = nonzero_ratio(clear)
# Function to get non-zero percentage and resized binary image
if white_ratio > 14:
    # if there is more than 14% of non-zero,
    gesture = predict(h_clear)
    # Function to predict and get gesture of the user
```

```
else:
    # if there is less than 14% of non-zero,
    gesture = 'Nothing'
    # assume there is no gesture given by the user
```

Importing necessary libraries.

```
import tensorflow as tf
import cv2
import numpy as np
```

Loading the trained CNN model that is used to predict gesture from a skin segmented binary image input.

```
model = tf.keras.models.load_model('CNN/binary_CNN_model.h5')
```

Get a cropped binary clear image of the gesture.

```
clear = crop_binary(frame)
# Function to get a cropped binary clear image
```

Get the percentage of white in the image and resized binary image.

```
white_ratio, h_clear = nonzero_ratio(clear)
# Function to get non-zero percentage and resized binary image
```

Predict gesture if percentage of white in the image is > 14%.

```
if white_ratio > 14:
    # if there is more than 14% of non-zero,
    gesture = predict(h_clear)
    # Function to predict and get gesture of the user
```

Gesture = ‘Nothing’ if percentage of white in the image is < 14%.

```
else:
    # if there is less than 14% of non-zero,
    gesture = 'Nothing'
    # assume there is no gesture given by the user
```

Integration and Testing

Folder Structure

To run the Scissor Paper Stone Game, several folders are created to organise the program.

Scissor Paper Stone Game

CNN

(Contains everything related to CNN model)

images

(Contains images used for training)

test

train

(Contain images used for validation and training of CNN model respectively)

binary_training.py

(Program to run the training of CNN model)

gathering_binary_image.py

(Program to take binary images for CNN model training)

binary_CNN_model.h5

(CNN model used for the game)

Music

(Contains music files and program to play the music during game)

Credits

Draw

Game

Instruction

Intro

Lose

Win

(Contain music file to play during the game at certain screen)

music.py

(Contains functions to load and play music during game)

Pictures

(Contains pictures used in the game and the program to load it)

Credits

Tutorial

(Contain pictures used for Credits and Instruction Screen)

background.png

hand.png

paper.png

scissors.png

stone.png

(Pictures for default background, for Scan screen, and for representing computer's gesture during the game respectively.)

pictures.py

(Script to load all pictures in the Pictures folder)

Sound

(Contains sound files and program to play the sound effect at certain event)

NFF-ethno-confirmation.wav

NFF-funny-cancel.wav

NFF-gong.wav

NFF-lose.wav

NFF-menu-04-a.wav

NFF-menu-a.wav

NFF-prompt-soft.wav

NFF-select-04.wav

(Sound files for the game)

sound.py

(Contains function to load and play sound)

Utils

(Contains function and values to be used frequently in the Main program)

color.py

(Script that contain variables stored with RGB values for font and shape colours used in the game)

utils.py

(Contain function that are used frequently)

main.py

(Main Program of the game)

Gameplay

(Please refer to the List of Functions section for details of the functions mentioned.)

Code to run the game (in main.py):

```
# Main
check_camera() # check if a camera is plugged in/available
```

To run, go into the Scissor Paper Stone folder in the Ubuntu terminal. Then type 'python3 main.py'.

(There is only one line of code in the Main

If no camera is plugged in, there will be a black screen with a prompt to the user to plug in camera will appear (by the check_camera() function from main.py):

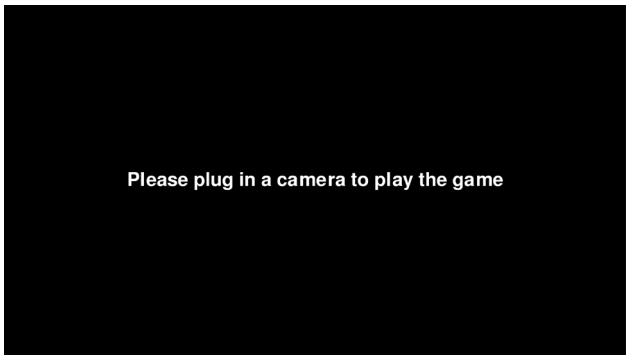


Figure 13 Prompt to plug in camera

If camera is plugged in afterwards, there will be a black screen with message that the program sensed the camera plugged in (by the camera_detected() function from main.py):

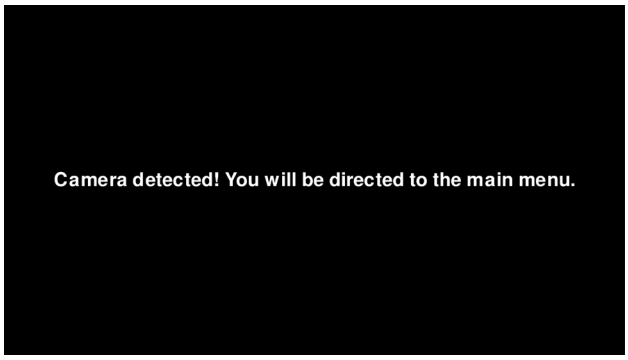


Figure 14 Message indicating camera has been detected by the program

After the message has been shown for about 4 seconds, the user will be directed to the Main Menu. (by camera_detected() function then, check_camera() function both from main.py)

The user will also see the Main Menu from the start if camera has been plugged in from the start. (by check_camera() function from main.py):

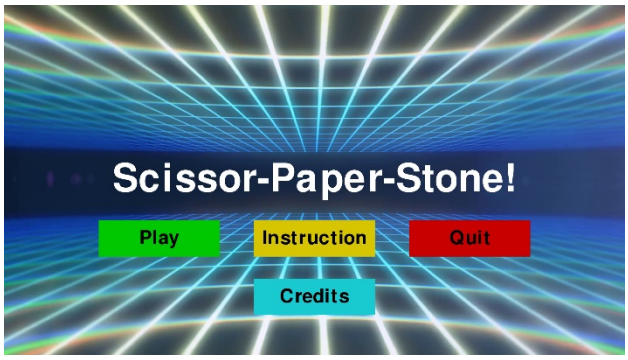


Figure 15 Main Menu

As seen in Figure 14, there are 4 buttons that can be pressed.

1. 'Play'
2. 'Instruction'
3. 'Credits'
4. 'Quit'

'Play'

Clicking the 'Play' button will direct the user to the Scan screen (by hand_scan_screen() function from main.py):

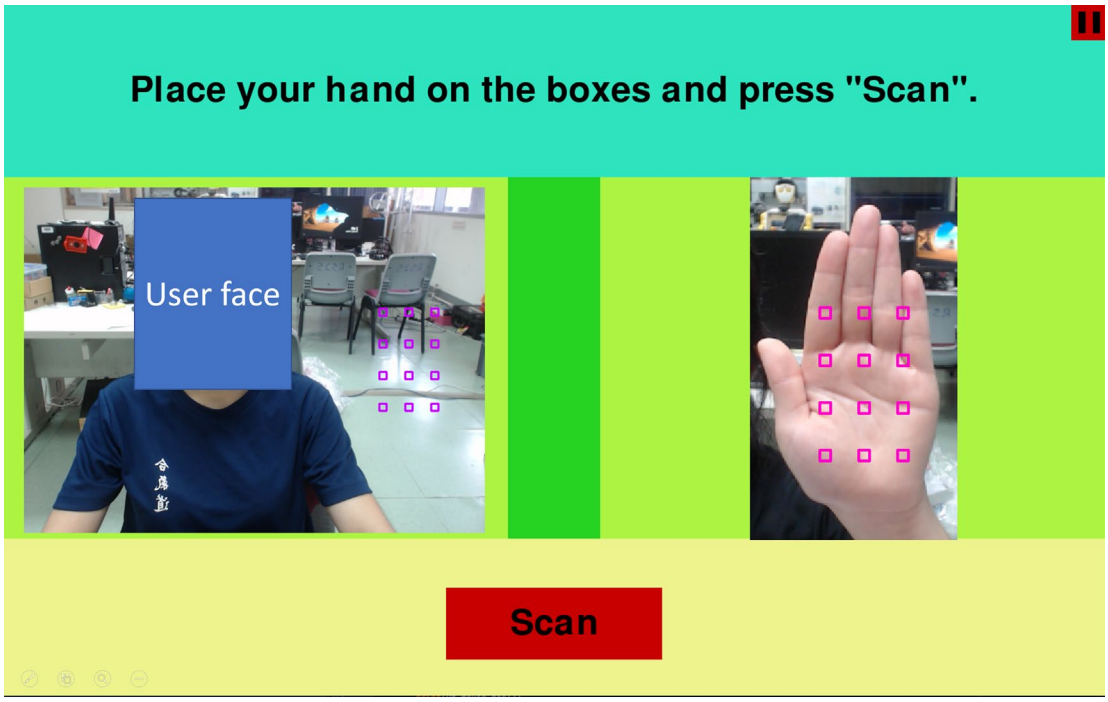


Figure 16 Scan screen

The user must position their hand such that it covers all the pink rectangles in the video stream like the picture on the right of the Scan screen.

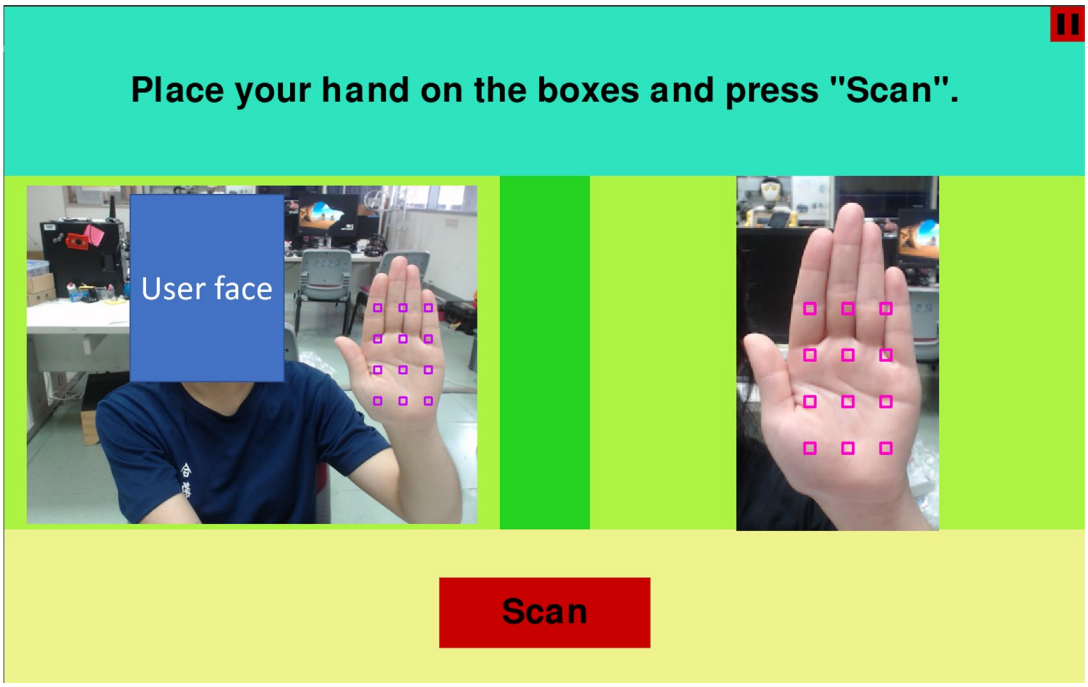


Figure 17 Scan screen with user position their hand at the squares

The user then clicks the 'Scan' button. When clicked, the hue and saturation histogram is created from the pixel bounded by the squares (by the hand_scan(frame) function from main.py). The histogram is stored in a variable called hand_hist.

The user is then directed to Game screen (by hand_scan_screen() function then game_screen() function both from main.py):

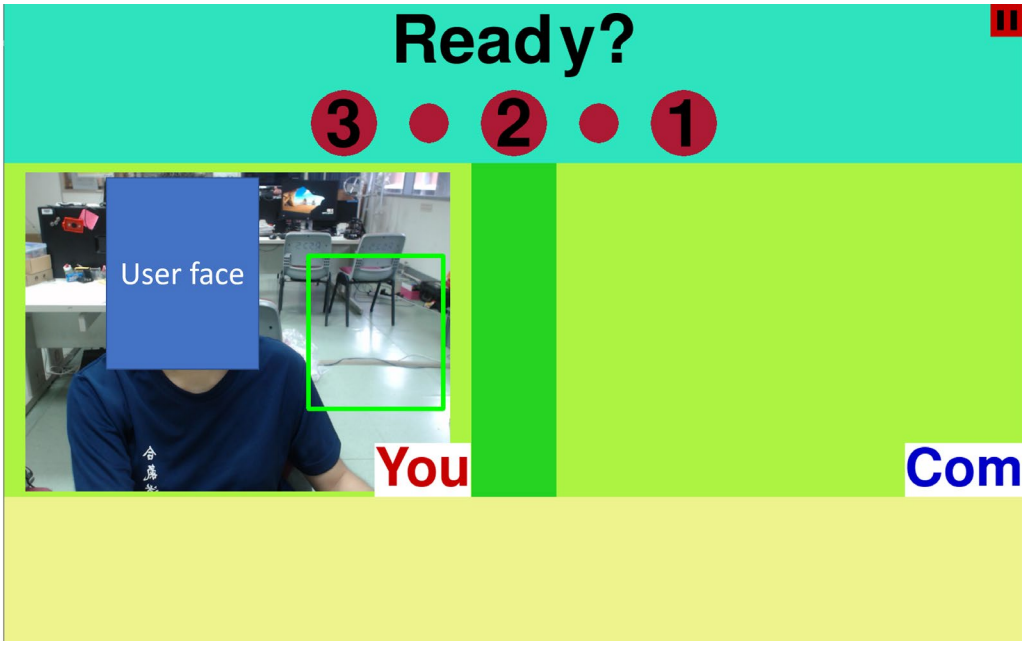


Figure 18 Game screen

After the word ‘Ready?’ (that inform the user to get ready to put a move) has been displayed for about 2 seconds, the game starts countdown. This will be indicated by the word and the circles displayed. (by SPS_timing(time) function from main.py)

The circles will change colour from red to orange from the most left circle with an interval of 0.5 seconds. The word will first change from ‘Ready?’ to ‘Scissor…’ (after the 2 seconds). Then it will change to ‘Paper…’ and finally ‘Stone!’ with an interval of 1 second. Pictures representing Scissor and Paper will also be displayed together with the word. (by SPS_timing(time) function from main.py)

For every change in word displayed, a sound effect is played to further indicate the change. (by SPS_timing(time) function from main.py)

Figure 19 Game screen with word 'Scissor...'

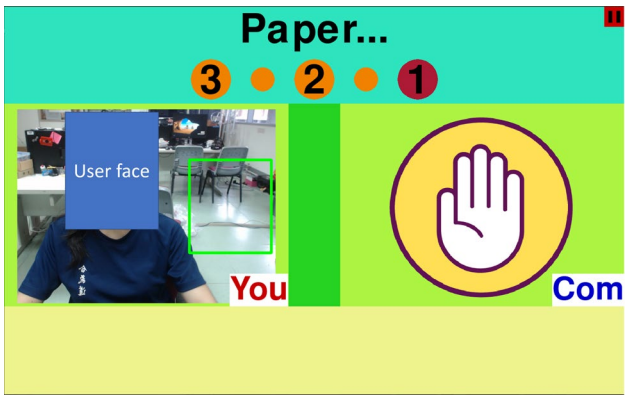


Figure 20 Game screen with word 'Paper...'

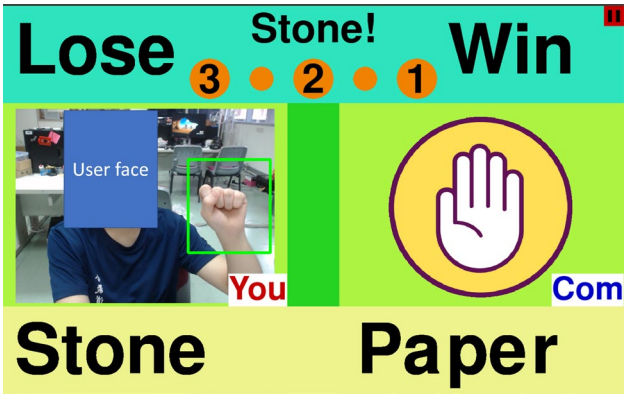


Figure 21 Game screen with word 'Stone!'

The user is to time when to put in their move into the green box using the words, circles and the sound from the game.

The user is to put their hand with the move into the box by the time the word has changed to ‘Stone!’ (like Figure 20).

When the word has changed to ‘Stone!’, the program crops the green rectangle area of the frame of the video stream. The cropped image is then back projected using the histogram (obtained earlier in the Scan screen) and is converted to binary image (where the skin colour is represented by white and others by black). The binary image is then processed to reduce noise in the image.
(above action by crop_binary(frame) from main.py)

The binary image is then resized to 100 x 100 pixels and the percentage of white in the image is calculated. (by nonzero_ratio(clear) from main.py)

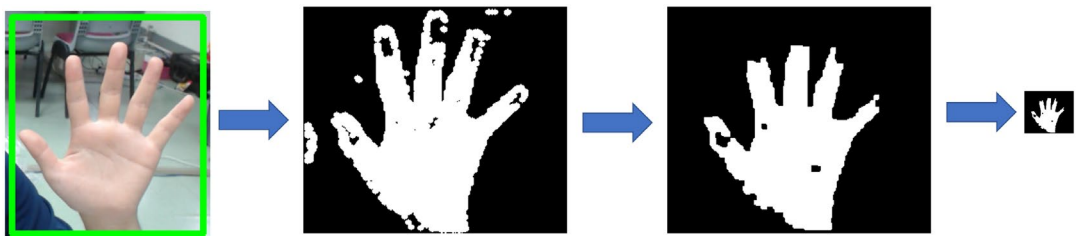


Figure 22 Process of processing frame

If the percentage of white is more than 14%, the binary image is fed to the CNN model for prediction. Else, the gesture is considered ‘Nothing’, meaning there is nothing in the cropped image.
(by detect(frame) function from main.py)
(For more details, please refer to ‘Prediction from CNN model’ in page 28)

The game then picks a random move from ‘Scissor’, ‘Paper’ and ‘Stone’ for the computer’s move (by random_gesture() function from main.py).

The user's predicted move and the computer's move are compared to decide who won and lose, or draw (by win_lose(gesture, rand_gesture) function from main.py).

The game then displays a screen with moves from both sides, the uncropped frame used for prediction, picture that represent computer's move and the result of the round for both sides (by display_gesture(gesture, rand_gesture, frame) function from main.py).
(Refer to Figure 21)

The game repeats from the 'Ready?' screen (refer to Figure 18) to the screen with the result of a round (refer to Figure 21) till 5 rounds in total is completed. In each round, a point is added to the winner or to both sides (for draw).
(by game_screen() function from main.py).

When all 5 rounds are completed, the total points for both sides are compared to decide who won or draw overall. The overall result is then shown (by result_screen(score_me, score_com from main.py):

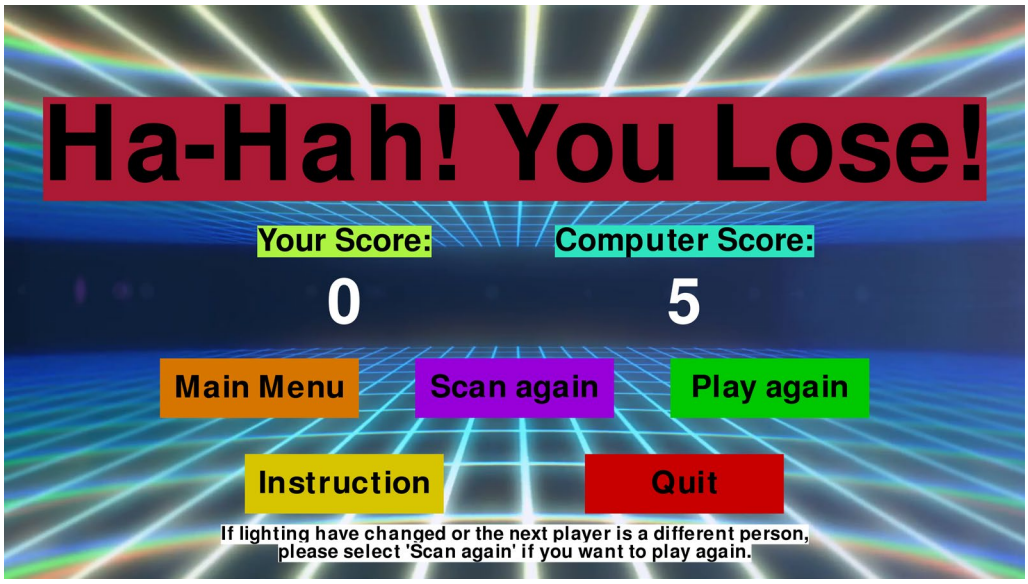


Figure 23 Overall Result screen

A message indicating whether the user has won, lose or draw will be displayed. Scores from both sides, buttons to navigate to other screens and a message (to prompt user to scan again if lighting has changed) will also be displayed.

If draw, the message at the top would be 'Draw? No fun here.' with yellow highlight.

Else, if the user win, the message is 'Tsk. You Win.' with green highlight.

Else, if the user loses, the message is 'Ha-Hah! You Lose!' with red highlight.

‘Instruction’

Instruction screens will be created by using pictures with instructions to fill the screen, together with 2 red buttons at the bottom of both sides to navigate through the Instruction screens and to exit to the Main Menu screen (when in first or last screen of Instruction).

(by tutorial_one(), tutorial_two(), tutorial_three(), tutorial_four(), tutorial_five(), tutorial_six(), tutorial_seven(), tutorial_eight(), tutorial_nine(), tutorial_ten(), tutorial_eleven(), tutorial_twelve(), tutorial_thirteen(), tutorial_fourteen(), tutorial_fifteen(), tutorial_sixteen(), tutorial_seventeen(), tutorial_eighteen(), tutorial_nineteen(), tutorial_twenty(), tutorial_twentyone() all from main.py)

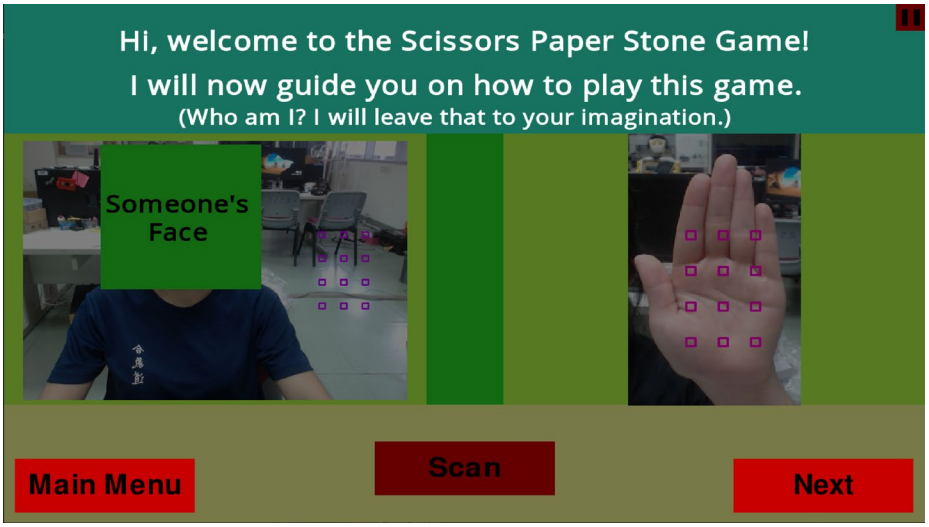


Figure 24 Instruction screen 1

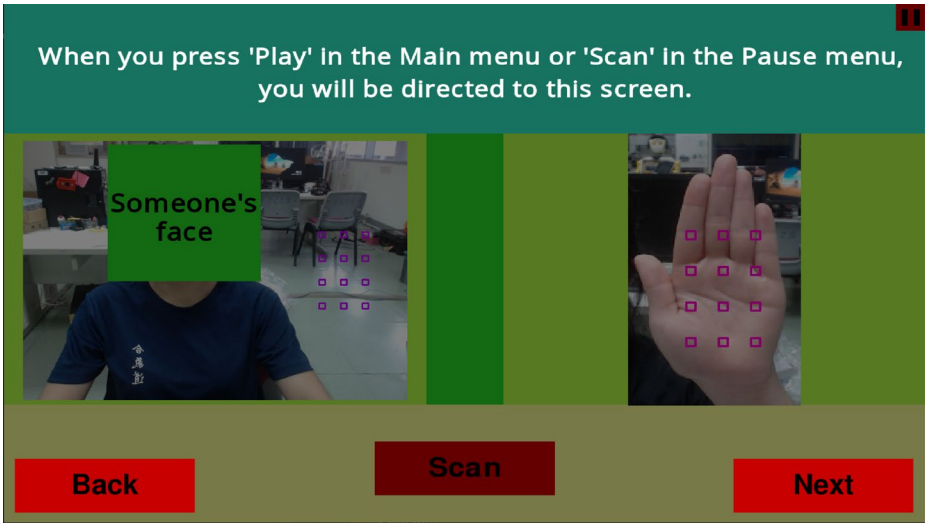


Figure 25 Instruction screen 2

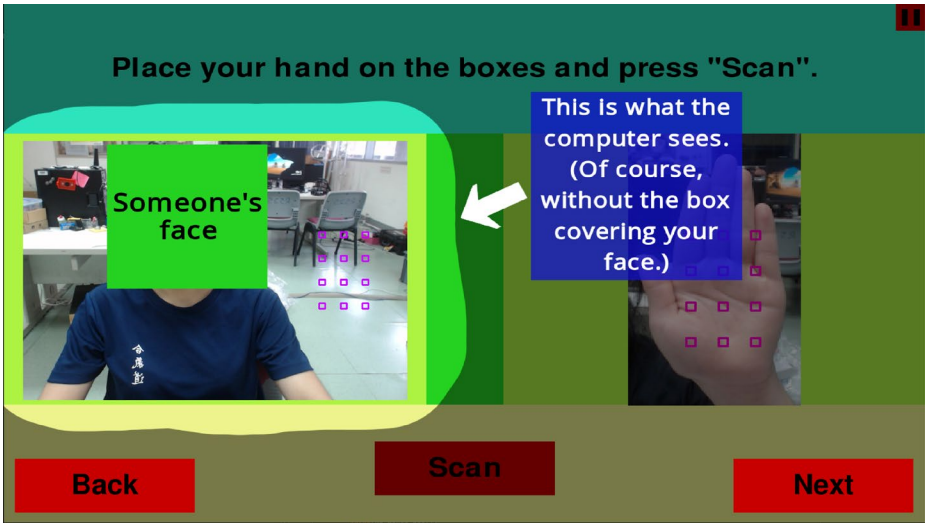


Figure 26 Instruction screen 3

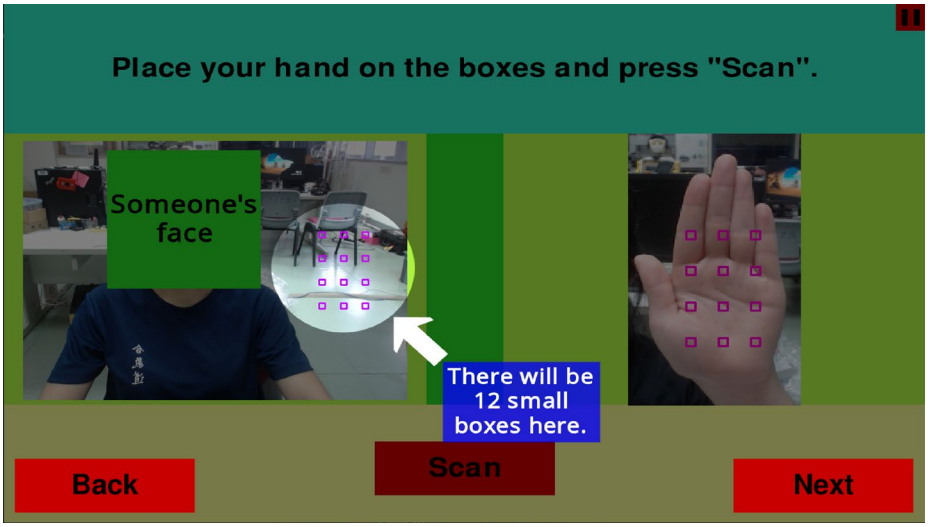


Figure 27 Instruction screen 4

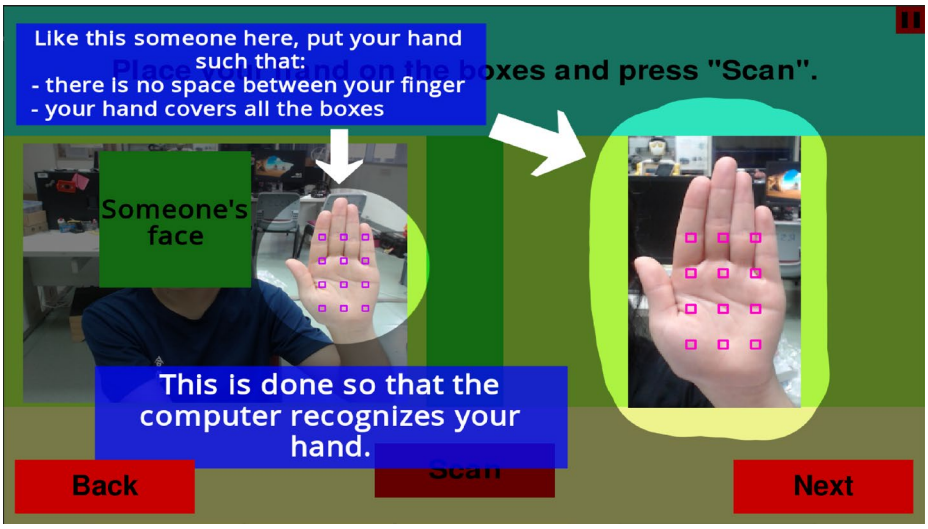


Figure 28 Instruction screen 5

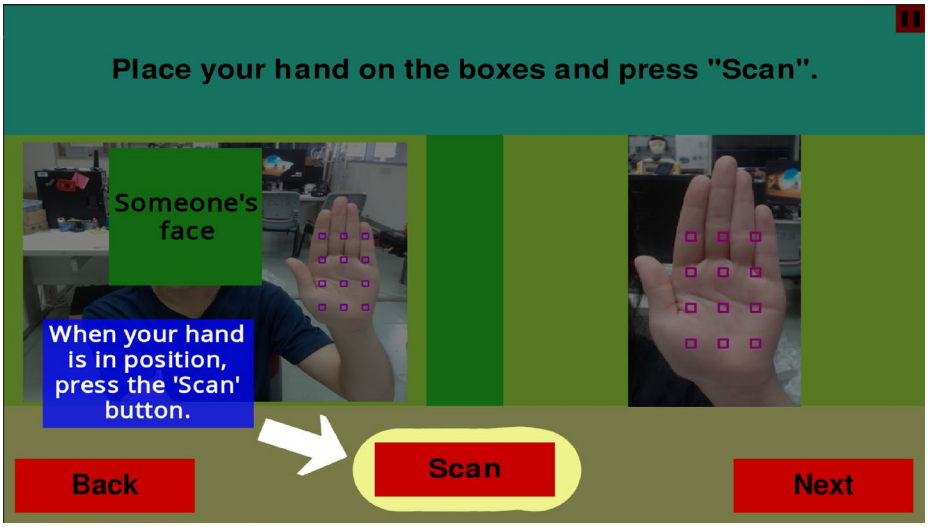


Figure 29 Instruction screen 6

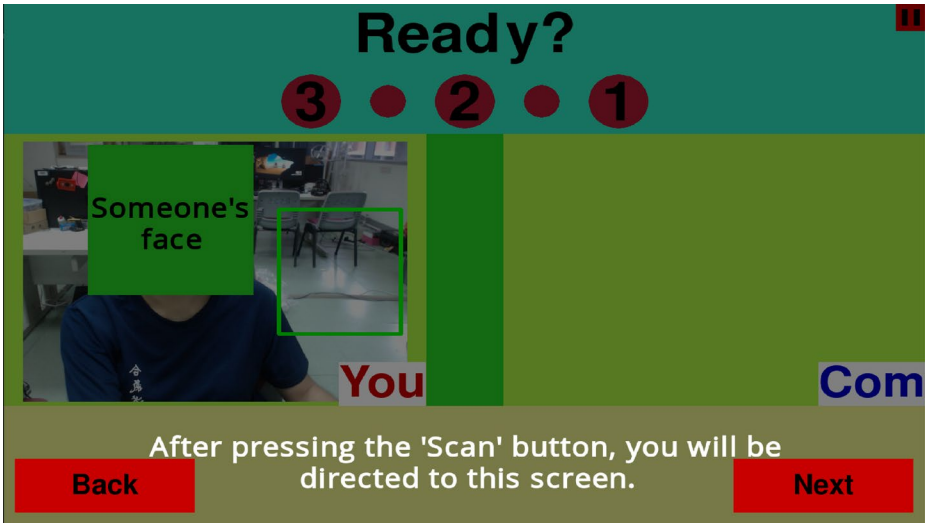


Figure 30 Instruction screen 7

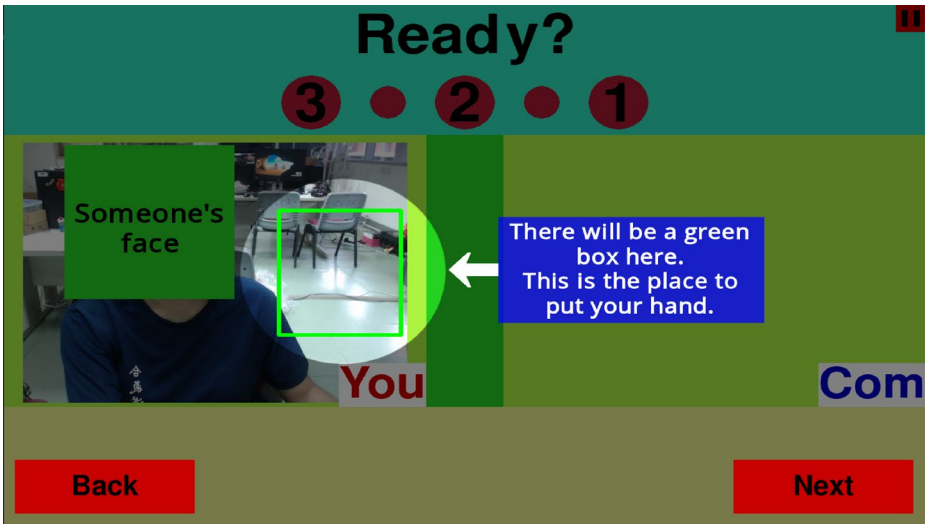


Figure 31 Instruction screen 8

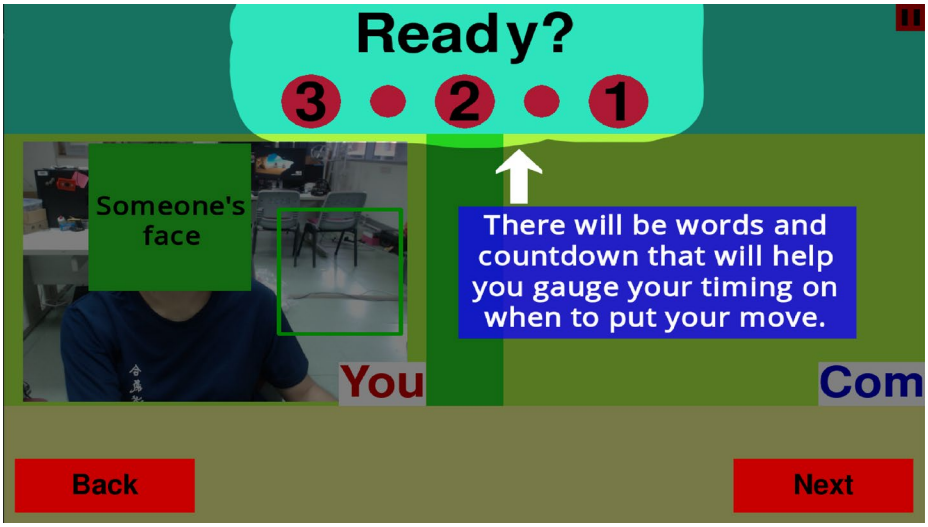


Figure 32 Instruction screen 9

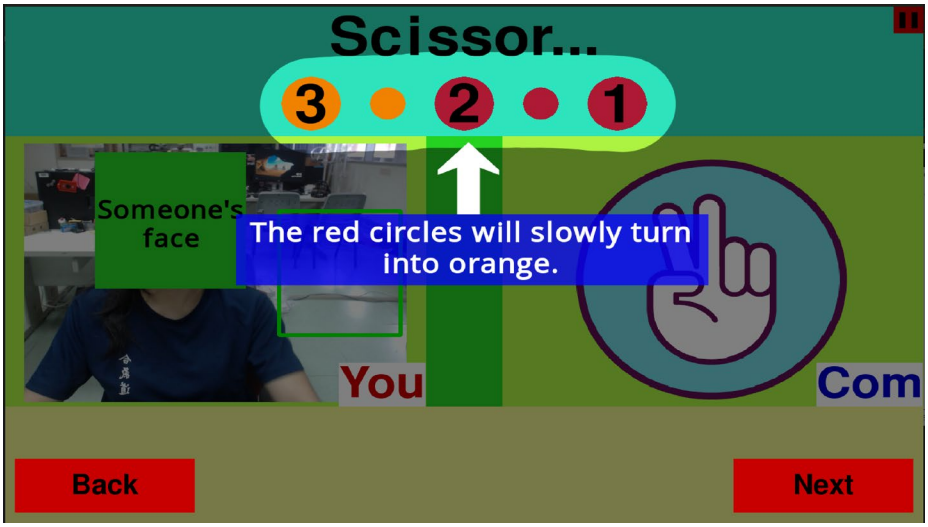


Figure 33 Instruction screen 10

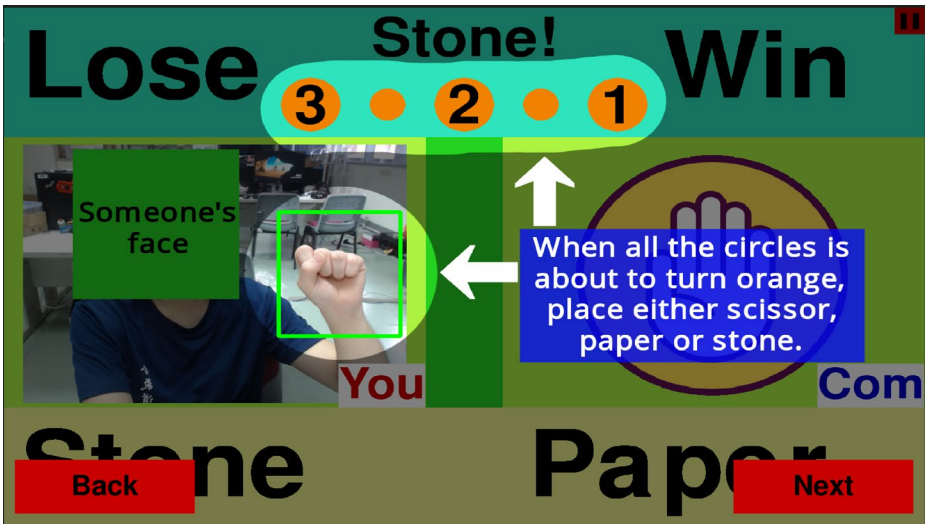


Figure 34 Instruction screen 11

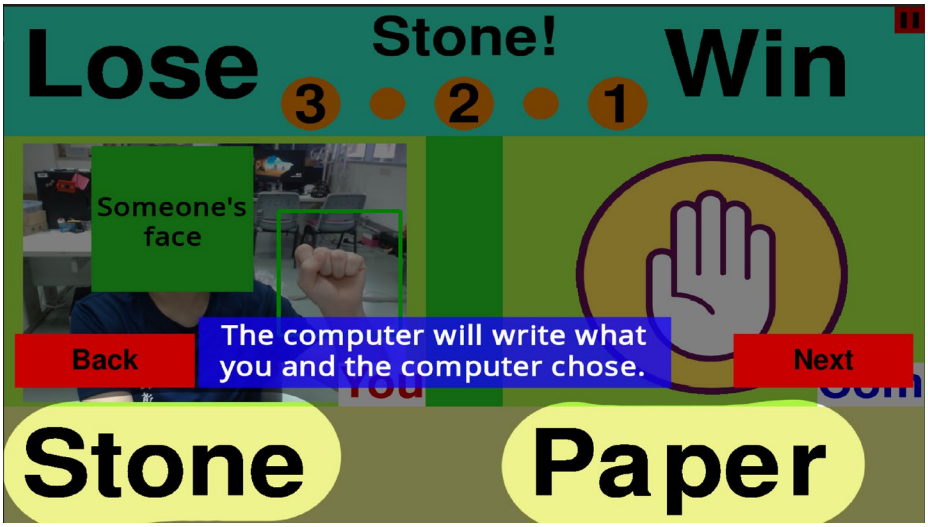


Figure 35 Instruction screen 12

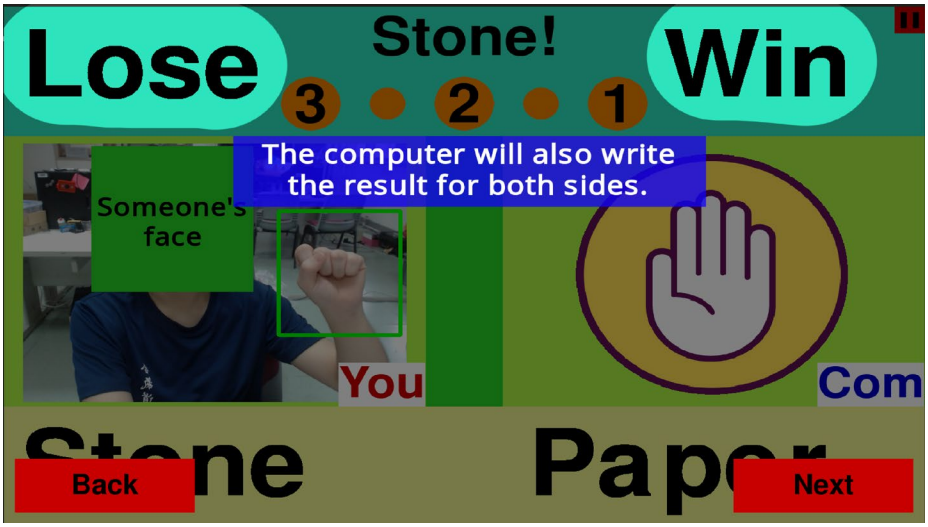


Figure 36 Instruction screen 13

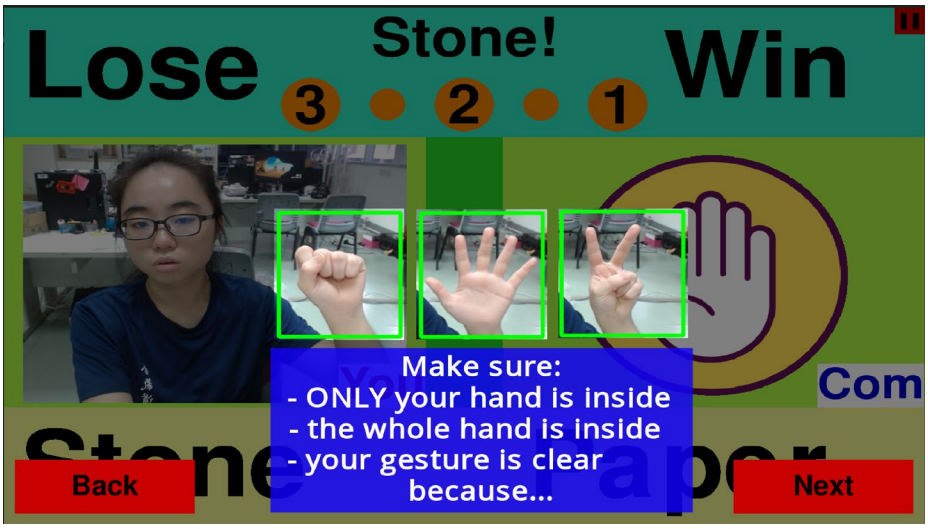


Figure 37 Instruction screen 14

Figure 38 Instruction screen 15

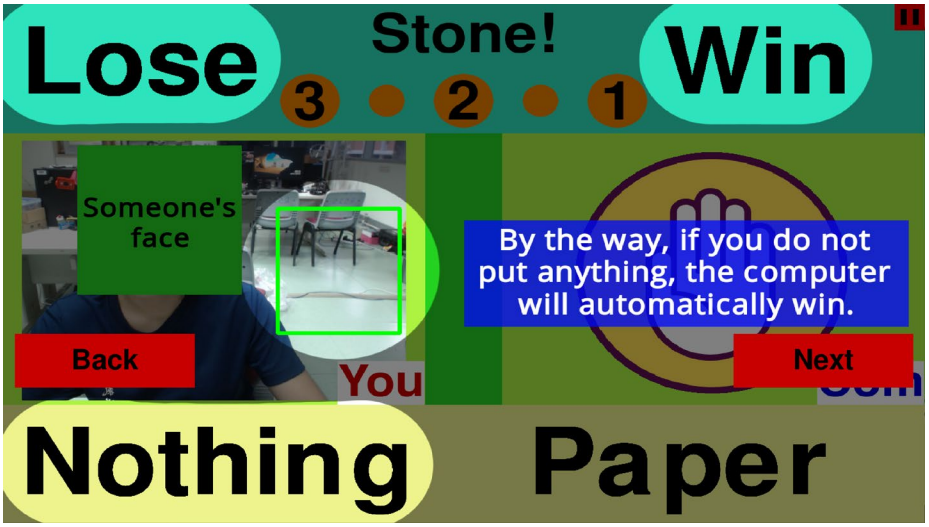


Figure 39 Instruction screen 16

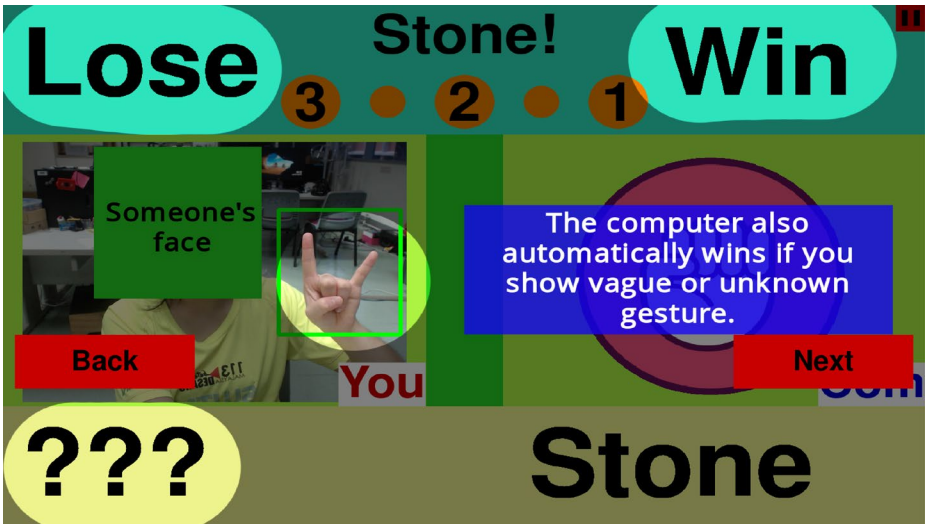


Figure 40 Instruction screen 17

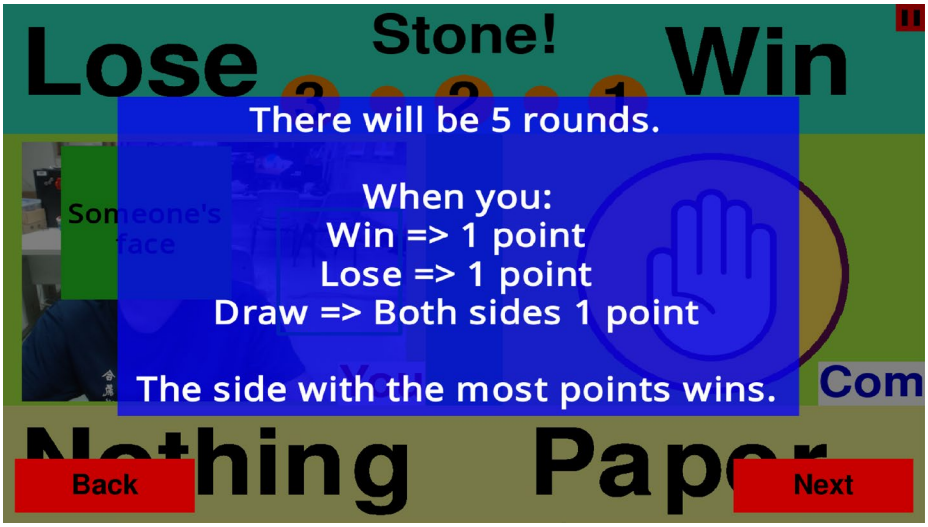


Figure 41 Instruction screen 18

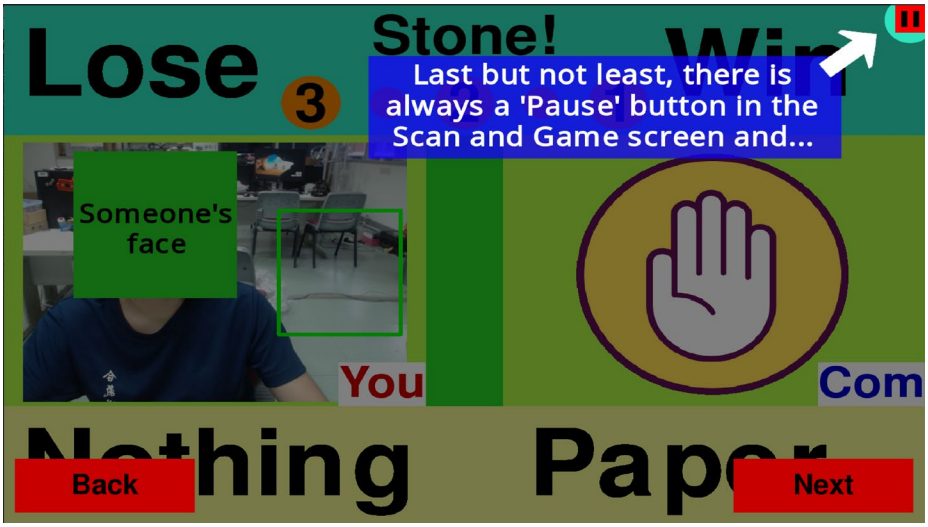


Figure 42 Instruction screen 19

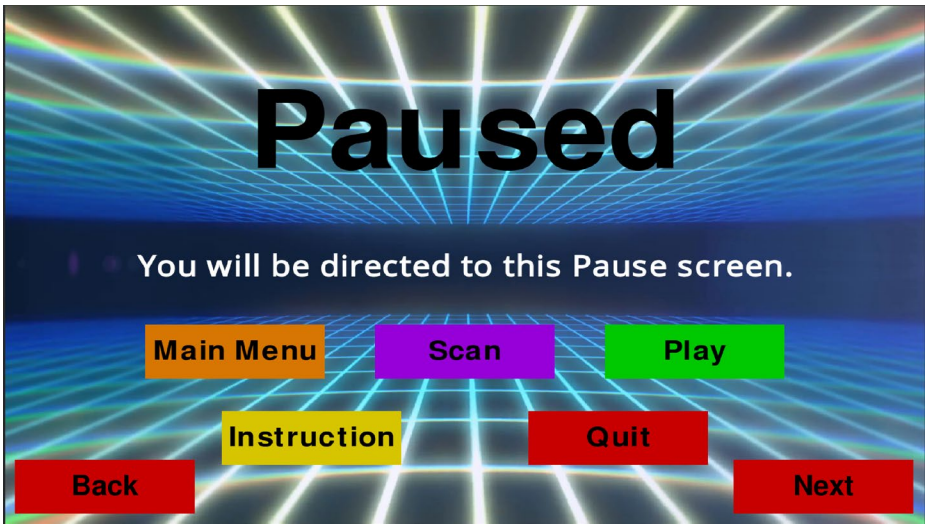


Figure 43 Instruction screen 20

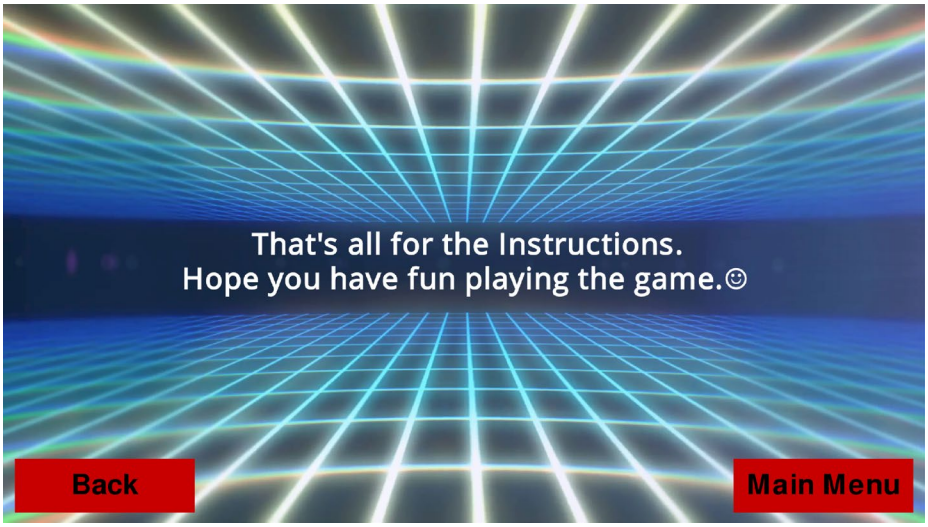


Figure 44 Instruction screen 21

'Credits'

Credits screens will be created by using pictures with credits to fill the screen, together with 2 red buttons at the bottom of both sides to navigate through the Credits screens and to exit to the Main Menu screen (when in first or last screen of Credits).

Figure 45 Credits screen 1

Figure 46 Credits screen 2

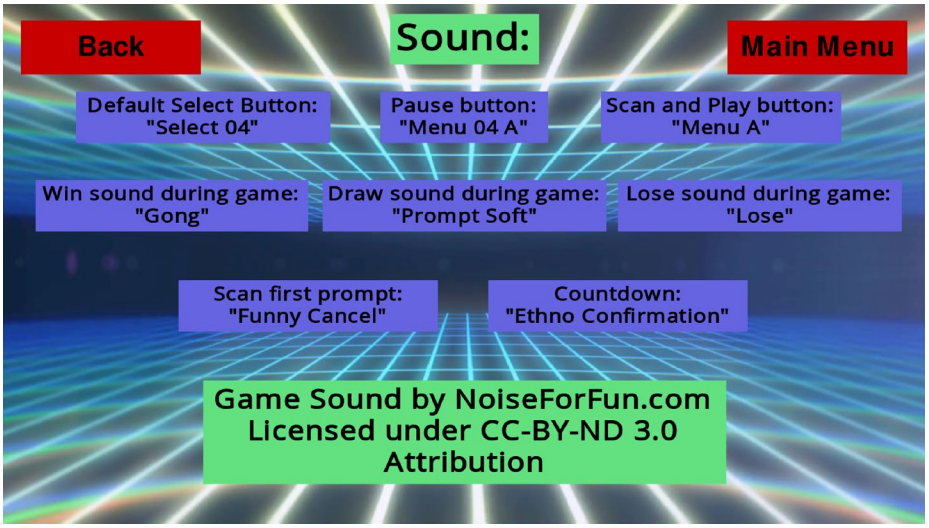


Figure 47 Credits screen 3

'Quit'

This button closes the game (by ut.quit_game() function from utils.py).

Key Technical Issues & Solutions

1. The game hangs while loading the webcam.
    a. **Description:**
        i. The game requires the webcam to be loaded for video capture in the Scan and Game screen. The loading of the webcam is done with cv2.VideoCapture() and the cursor will be stalled at that line of code until the webcam is loaded.
    b. **Reason:**
        i. The reason is not identified but the loading of the webcam is slow on Windows 10 but immediate Ubuntu OS (16.04 & 18.04). The difference in the OS may have caused the delay in the loading.
    c. **Solution:**
        i. Instead of using Windows 10, Ubuntu 18.04 is used to run the game.
2. In the game, music stops once the screen has been changed when not programmed to.
    a. **Description:**
        i. After pressing the button to go into a different screen in the game, the music suddenly stops, and the button sound plays. Afterwards, the music did not start playing.
    b. **Reason:**
        i. The music and the button sound are playing on the same channel hence the button sound has replaced the music when a button is pressed.
    c. **Solution:**
        i. Instead of using pygame.mixer.Sound.play(pressing_sound) to play the sound, I used pygame.mixer.Channel(0).play(pygame.mixer.Sound(pressing_sound)) to play the sound in another channel different from the channel where the music is playing.
3. There was a way to go into the Game screen without 'Scanning' user's hand.
    a. **Description:**
        i. In the Scan screen without pressing the 'Scan' button, press the Pause button at the top right corner. After that, press 'Play' button. The Game screen appears and starts the countdown. After the countdown is over, the game crashes as there is no histogram to use to create the skin segmented binary image.
    b. **Reason:**
        i. I have not set condition to go into the Game screen that restricts go into it without 'Scanning' the hand.
    c. **Solution:**
        i. I have made a global variable, 'scanned', to know whether there was any scanning done. When the hand is scanned, the 'scanned' variable (that was set to 0 upon entering the Scan screen) is changed to integer value of 1. Using this, when 'Play' button is pressed in the Pause screen (from Scan screen) without any scanning done, the button function will check for the value of the 'scanned'. If 'scanned' equals to 1, the user will be directed to

the Game screen. Else, there will be a message in the middle of the Pause screen to inform user that they have to scan their hand first.

4. The number of points for both sides is reset when the Pause screen is accessed in the middle of playing the game in the Game screen.
    a. **Description:**
        i. For example, the user and the computer have 1 point each after playing two rounds of the game. The user then accesses the Pause screen and the scores for both sides are reset to zero.
    b. **Reason:**
        i. The two variables that keeps the score for both sides are local variables and are in the function that creates the Game screen. The Pause screen is created with another different function. Hence when going into the Game screen function again after the Pause screen function, a completely new variable is created with the same name.
    c. **Solution:**
        i. I made those two variables global so that the Game screen function can keep track of the scores.

Full Code

(Please refer to the next section, 'List of Functions', for the code of the functions.)

Main Program (main.py)

```
import pygame
import time
import cv2
import numpy as np
import time
import tensorflow as tf
import os
import random
from Music import music
from Sound import sound
from Utils import color as c
from Utils import utils as ut
from Pictures import pictures as pic
import math as m

##########################################################################################
# Load CNN model for predicting gestures
model = tf.keras.models.load_model('CNN/binary_CNN_model.h5')
##########################################################################################


##########################################################################################
# Initializing for sound, music and pygame itself
pygame.mixer.pre_init(44100, -16, 2, 512)
pygame.init()
##########################################################################################


##########################################################################################
# Making the window fullscreen
gameDisplay = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
# Getting the dimension of the window size
display_width, display_height = pygame.display.get_surface().get_size()
# Setting the window caption
pygame.display.set_caption('Scissor Paper Stone!')
##########################################################################################


##########################################################################################
# Setting constants for storing certain states and calculation
matches = int(0)
init_sound = 0
PI = 3.14159
display_area = display_width * display_height
score_me = int(0)
score_com = int(0)
##########################################################################################
```

```
##############################################################################
# Setting font size of title in Main menu
title_size = int((display_area*0.007)**(1/2))
##############################################################################


##############################################################################
# Setting button font size and button dimension
button_font_size = int(m.sqrt(display_area*0.0017))
button_w = int(display_width*0.19531)
button_h = int(display_height*0.10417)
##############################################################################


##############################################################################
# Setting default tutorial button ('Back', 'Next') placement offset
tutt_button_offset_x = int(display_width*0.01302)
tutt_button_offset_y = int(display_height*0.02083)
# Setting tutorial button ('Back', 'Next') y position when button is raised
tutt_raised_button_y_pos = int(display_height*0.63542)
##############################################################################


##############################################################################
# Setting default offset for placing button and text
offset_x = int(display_width * 0.01953)
offset_y = int(display_height * 0.03125)
##############################################################################


##############################################################################
# Setting Pause button dimension and the black bars' dimension and placemen
paused_text_size = int(m.sqrt(display_area*0.02713))
pause_button_size = int(m.sqrt(display_area*0.0017))
black_bar1_x, black_bar1_y, black_bar1_w, black_bar1_h = \
    display_width - int(pause_button_size*0.8), int(pause_button_size*0.2),
    int(pause_button_size*0.2), int(pause_button_size*0.6)
black_bar2_x, black_bar2_y, black_bar2_w, black_bar2_h = \
    display_width - int(pause_button_size*0.4), int(pause_button_size*0.2),
    int(pause_button_size*0.2), int(pause_button_size*0.6)
##############################################################################


##############################################################################
# Setting the computer gesture's placement
gesture_w, gesture_h = pic.scissors.get_rect().size
gesture_pic_x = display_width // 2 + display_width // 24 + \
                (display_width // 2 - display_width // 24 - gesture_w) // 2
gesture_pic_y = display_height // 4 + (display_height // 2 - gesture_h)//2
##############################################################################
```

```
##############################################################################
# Setting the font size for result ('Win', Lose', Draw') and
# gesture ('Scissor', 'Paper', 'Stone', 'Nothing', '???')
result_gesture_text = int(m.sqrt(display_area*0.01736))
##############################################################################


##############################################################################
# Setting font size for 'You' and 'Com'
you_com_text = int(m.sqrt(display_area*0.00434))
##############################################################################


##############################################################################
# Setting the clock for FPS
clock = pygame.time.Clock()
##############################################################################


##############################################################################
# Function to check whether camera is plugged in/available and
# show message to inform the user of the situation if necessary
def check_camera():...

def camera_detected():...
##############################################################################


##############################################################################
# Functions to create and display text
def text_object(msg, font, color, back_color=None):...

def text(msg, text_size, color, x, y, positioning = '', back_color=None,
         font_type='freesansbold.ttf'):...

# Function to get the width and the height of the text
def text_size(msg, font_size, font_type='freesansbold.ttf'):...
##############################################################################


##############################################################################
# Function to create text for button
def button_text(msg, text_size, color, font_type, x, y, w, h):...

# Function to create button with text
def button(placement, x, y, w, h, default_color, light_up_color, msg, text_size,
           text_color, previous, action=None, font_type='freesansbold.ttf'):...
```

```
# Function to create pause button
def pause_button():...
##############################################################################


##############################################################################
# Functions to create Instruction screens
def tutorial_one():...
def tutorial_two():...
def tutorial_three():...
def tutorial_four():...
def tutorial_five():...
def tutorial_six():...
def tutorial_seven():...
def tutorial_eight():...
def tutorial_nine():...
def tutorial_ten():...
def tutorial_eleven():...
def tutorial_twelve():...
def tutorial_thirteen():...
def tutorial_fourteen():...
def tutorial_fifteen():...
def tutorial_sixteen():...
def tutorial_seventeen():...
def tutorial_eighteen():...
def tutorial_nineteen():...
def tutorial_twenty():...
def tutorial_twentyone():...
##############################################################################


##############################################################################
# Function to create Main menu screen
def intro():...
##############################################################################


##############################################################################
# Functions to create Credits screens
def credits_one():...

def credits_two():...

def credits_three():...
##############################################################################
```

```
##########################################################################################
# Function to display the Screen/Game background
def init_game_screen(run=True):...
##########################################################################################


##########################################################################################
# Function to display text ('You' and 'Com') in the scan/game screen
def you_com():...

# Function to display words ('Ready', 'Scissor', 'Paper', 'Stone'),
# circles with text and changing colors and sound for timing the move
def SPS_timing(time):...
##########################################################################################


##########################################################################################
# Function to create Pause menu
def pause_menu():...

# Function to create Pause menu with message to tell the user to scan the hand first
def pause_scan_first():...
##########################################################################################


##########################################################################################
# Function to draw reddish pink rectangles on the copy of the frame and
# combine the copy and the frame
def translucent_obj(frame, action=None, alpha=0.5):...

# Function to calculate and get the hand histogram
def set_hand_hist(frame, start_rect, end_rect):...

# Function to draw reddish pink rectangles to scan hand
def draw_rect_right(frame, return_list_only=False, color=(200, 0, 255),thickness=2):...

# Function to get hand histogram
def hand_scan(frame):...

# Function to make the cropped frame into binary image
def hist_mask(frame, hand_hist):...

# Function to make binary image clearer
def clearer(frame, kernel_size=5, iteration=1):...

# Function to prepare images to feed into the CNN for prediction
def crop_binary(frame):...

# Function to put in video in the Scan or Game screen
def frame_adjust(frame, hori_1, hori_2, vert_1, vert_2, scan_now=False):...
```

```
# Function to create Scan screen
def hand_scan_screen():...
##############################################################################################


##############################################################################################
# Function to get the percentage of non-zero in the cropped
# binary image and get the cropped binary image resized to 100 x 100
def nonzero_ratio(clear):...

# Function to predict the gesture in the image using CNN
def predict(h_clear):...

# Function to predict gesture when necessary
def detect(frame):...

# Function to randomly choose a gesture ('Scissor', 'Paper', 'Stone') for the computer
def random_gesture():...

# Function to get the result of the match for both user and computer
def win_lose(gesture, rand_gesture):...

# Function to display game result screen for 2 seconds
def display_gesture(gesture, rand_gesture, frame):...

# Function to create Game screen
def game_screen():...
##############################################################################################


##############################################################################################
# Function to show result screen
def result_screen(score_me, score_com):...
##############################################################################################


# Main
check_camera() # check if a camera is plugged in/available
```

Music Program (music.py)

```
import pygame
import os
import random

# (below) get and store names of music from folders into
# their respective list variables
Intro = os.listdir('Music/Intro')
Instruction = os.listdir('Music/Instruction')
Game = os.listdir('Music/Game')
Win = os.listdir('Music/Win')
Lose = os.listdir('Music/Lose')
Draw = os.listdir('Music/Draw')
Credits = os.listdir('Music/Credits')

def intro():...

def instruc():...

def game():...

def win():...

def lose():...

def draw():...

def credits():...

def stop():...
```

Sound Effect Program (sound.py)

```
import pygame

def select():...

def pause():...

def play():...

def scan_first():...

def win():...

def lose():...

def draw():...

def timing():...
```

Utilities Program (utils.py)

```
import pygame

def mouse_pos():...

def update_screen(clock):...

def quit_x():...

def quit_game():...
```

Pictures Script (pictures.py)

```
import pygame
import math as m

pygame.init() # initialize pygame

###############################################################################
# (below) get size of the display size
display = pygame.display.Info()
display_width = int(display.current_w)
display_height = int(display.current_h)
###############################################################################


###############################################################################
# Calculating display area for calculation
display_area = display_width * display_height

# Setting constants for the dimension of the image of the hand in the scan scree
hand_h = int(display_height // 2 - display_width*0.0625)
hand_w = int(hand_h*0.57143)

# Setting constants for the dimension of the image of the computer's gesture
sps_move = int(m.sqrt(display_area*0.156)+m.sqrt(display_area*0.156)*0.0008)
###############################################################################


###############################################################################
# Loading and resizing default background image
background = pygame.image.load('Pictures/background.png')
background = pygame.transform.scale(background, (display_width, display_height))

# Loading and resizing image of hand in the scan screen
hand = pygame.image.load('Pictures/hand.png')
hand = pygame.transform.scale(hand, (hand_w, hand_h))

# Loading and resizing images of scissors, paper and stone to represent computer's gesture
scissors = pygame.image.load('Pictures/scissors.png')
scissors = pygame.transform.scale(scissors, (sps_move, sps_move))
paper = pygame.image.load('Pictures/paper.png')
paper = pygame.transform.scale(paper, (sps_move, sps_move))
stone = pygame.image.load('Pictures/stone.png')
stone = pygame.transform.scale(stone, (sps_move, sps_move))
```

```
# Loading and resizing images for Instructions
tutorial_1 = pygame.image.load('Pictures/Tutorial/tutorial_hi.jpg')
tutorial_2 = pygame.image.load('Pictures/Tutorial/tutorial_directed_to.jpg')
tutorial_3 = pygame.image.load('Pictures/Tutorial/tutorial_com_see.jpg')
tutorial_4 = pygame.image.load('Pictures/Tutorial/tutorial_box.jpg')
tutorial_5 = pygame.image.load('Pictures/Tutorial/tutorial_put_hand.jpg')
tutorial_6 = pygame.image.load('Pictures/Tutorial/tutorial_press_scan.jpg')
tutorial_7 = pygame.image.load('Pictures/Tutorial/tutorial_game_screen.jpg')
tutorial_8 = pygame.image.load('Pictures/Tutorial/tutorial_green_box.jpg')
tutorial_9 = pygame.image.load('Pictures/Tutorial/tutorial_gauge.jpg')
tutorial_10 = pygame.image.load('Pictures/Tutorial/tutorial_red_circles.jpg')
tutorial_11 = pygame.image.load('Pictures/Tutorial/tutorial_when_orange.jpg')
tutorial_12 = pygame.image.load('Pictures/Tutorial/tutorial_chose_what.jpg')
tutorial_13 = pygame.image.load('Pictures/Tutorial/tutorial_result.jpg')
tutorial_14 = pygame.image.load('Pictures/Tutorial/tutorial_gesture_clear.jpg')
tutorial_15 = pygame.image.load('Pictures/Tutorial/tutorial_something_like_this.jpg')
tutorial_16 = pygame.image.load('Pictures/Tutorial/tutorial_nothing.jpg')
tutorial_17 = pygame.image.load('Pictures/Tutorial/tutorial_unknown.jpg')
tutorial_18 = pygame.image.load('Pictures/Tutorial/tutorial_points.jpg')
tutorial_19 = pygame.image.load('Pictures/Tutorial/tutorial_pause_button.jpg')
tutorial_20 = pygame.image.load('Pictures/Tutorial/tutorial_pause_menu.jpg')
tutorial_21 = pygame.image.load('Pictures/Tutorial/tutorial_thanks.jpg')
tutorial_1 = pygame.transform.scale(tutorial_1, (display_width, display_height))
tutorial_2 = pygame.transform.scale(tutorial_2, (display_width, display_height))
tutorial_3 = pygame.transform.scale(tutorial_3, (display_width, display_height))
tutorial_4 = pygame.transform.scale(tutorial_4, (display_width, display_height))
tutorial_5 = pygame.transform.scale(tutorial_5, (display_width, display_height))
tutorial_6 = pygame.transform.scale(tutorial_6, (display_width, display_height))
tutorial_7 = pygame.transform.scale(tutorial_7, (display_width, display_height))
tutorial_8 = pygame.transform.scale(tutorial_8, (display_width, display_height))
tutorial_9 = pygame.transform.scale(tutorial_9, (display_width, display_height))
tutorial_10 = pygame.transform.scale(tutorial_10, (display_width, display_height))
tutorial_11 = pygame.transform.scale(tutorial_11, (display_width, display_height))
tutorial_12 = pygame.transform.scale(tutorial_12, (display_width, display_height))
tutorial_13 = pygame.transform.scale(tutorial_13, (display_width, display_height))
tutorial_14 = pygame.transform.scale(tutorial_14, (display_width, display_height))
tutorial_15 = pygame.transform.scale(tutorial_15, (display_width, display_height))
tutorial_16 = pygame.transform.scale(tutorial_16, (display_width, display_height))
tutorial_17 = pygame.transform.scale(tutorial_17, (display_width, display_height))
tutorial_18 = pygame.transform.scale(tutorial_18, (display_width, display_height))
tutorial_19 = pygame.transform.scale(tutorial_19, (display_width, display_height))
tutorial_20 = pygame.transform.scale(tutorial_20, (display_width, display_height))
tutorial_21 = pygame.transform.scale(tutorial_21, (display_width, display_height))

# Loading and resizing images for credits
credits_1 = pygame.image.load('Pictures/Credits/credits_pictures.jpg')
credits_2 = pygame.image.load('Pictures/Credits/credits_music.jpg')
credits_3 = pygame.image.load('Pictures/Credits/credits_sound.jpg')
credits_1 = pygame.transform.scale(credits_1, (display_width, display_height))
credits_2 = pygame.transform.scale(credits_2, (display_width, display_height))
credits_3 = pygame.transform.scale(credits_3, (display_width, display_height))
################################################################################
```

List of Functions

Main program (main.py)

Camera availability check

- **Main**
  a. check_camera()

```
# Function to check whether camera is plugged in/available and
# show message to inform the user of the situation if necessary
def check_camera():
    ''' To check if camera is plugged in/available '''

    cam_plug_first = True
    # Bool variable to check if camera is plugged in/available at first.
    # (above)   If cam_plug_first = False, it means camera wasn't plugged at first.

    message_size = int(m.sqrt(display_area*0.0017))
    # Setting font size of message

    while True:
        cap = cv2.VideoCapture(0)
        if not cap.isOpened():
            # If there is no camera is plugged in/available
            gameDisplay.fill(c.black)
            # Filling the screen with black color
            text('Please plug in a camera to play the game', message_size, c.white,
                 display_width, display_height, 'center of box')
            # (above)  Displaying message in the middle to ask user to plug in a camera
            ut.update_screen(clock)
            # Updating screen
            cam_plug_first = False
            # Indicating camera wasn't plugged in/available
            ut.quit_x()
            # Close window if 'X' button (top right) or ESC key is pressed
        else:
            # If there is a camera plugged in/available
            if not cam_plug_first:
                # If camera wasn't plugged in/available at first,
                camera_detected()
                # Show message to inform user that camera has been plugged in
                break
                # Break out of the loop to go into the Main Menu
            else:
                # If camera was plugged in/available at first
                break
                # Break out of the loop to go into the Main Menu
    cap.release()
    # Release the camera for capturing frames
    intro()
    # Go to the Main Menu
```

- **Display**
  a. camera_detected()

```
def camera_detected():
    '''To display message of camera being detected for about 4 seconds'''
    start = time.time() # To start timing the duration of the message
    message_size = int(m.sqrt(display_area*0.0017)) # Setting font size of message
    while time.time() - start < 4: # while the duration of the message is below 4 seconds
        gameDisplay.fill(c.black) # Filling the screen with black color
        text('Camera detected! You will be directed to the main menu.', message_size, c.white,
             display_width, display_height, 'center of box')
        # (above)  Displaying message in the middle to let user know of the situation
        ut.update_screen(clock) # Updating screen
        ut.quit_x() # Close window if 'X' button (top right) or ESC key is pressed
```

Main Menu

- **Main**
  a. intro()

```
# Function to create Main menu screen
def intro():
    '''Create Main menu with buttons, music and title'''
    music.stop() # stop playing the music previously played
    music.intro() # play music for Main menu
    while True:
        gameDisplay.blit(pic.background, (0, 0)) # display default background

        text('Scissor-Paper-Stone!', title_size, c.white, display_width, display_height,
             'center of box')
        # display title for the Main menu
        button('center', display_width//4, display_height//3*2, button_w, button_h,
               c.green_dim, c.green_lit, 'Play', button_font_size, c.black, 'intro',
               'scan_screen')
        # (above) creating button to go into Scan screen
        button('center', display_width//4 * 2, display_height//3*2, button_w, button_h,
               c.yellow_dim, c.yellow_lit, 'Instruction', button_font_size, c.black,
               'intro', 'tutt_1')
        # (above) creating button to go into Tutorial 1
        button('center', display_width//4 * 3, display_height//3*2, button_w, button_h,
               c.red_dim, c.red_lit, 'Quit', button_font_size, c.black, 'intro', 'quit')
        # (above) creating button to close the window
        button('center', display_width // 4 * 2, display_height // 6 * 5, button_w,
               button_h, c.light_blue_dim, c.light_blue_lit, 'Credits', button_font_size,
               c.black, 'intro', 'credits_one')
        # (above) creating button to go into Credits 1 screen
        ut.update_screen(clock) # Updating screen
        ut.quit_x() # Close window if 'X' button (top right) or ESC key is pressed
```

Scan Screen

- **Main**
    a. hand_scan_screen()

```
# Function to create Scan screen
def hand_scan_screen():
    '''Create Scan screen with Scan button'''
    global hand_hist
    # get the extracted hand color histogram
    global scanned
    # get the scanned var. to store the state of whether the hand has been scanned
    global cap
    # var. to get the video capture
    hand_hist = None
    # reset the hand histogram
    scanned = 0
    # reset the scanned state
    cap = cv2.VideoCapture(0)
    # start taking video
    while cap.isOpened():
        # do while loop if video is properly taken
        _, frame = cap.read()
        # get the frame of the video
        hori_1, hori_2, vert_1, vert_2 = init_game_screen(True)
        # get the frame position and draw the game default background
        frame_adjust(frame, hori_1, hori_2, vert_1, vert_2, True)
        # postition the frame in the scan screen

        text('Place your hand on the boxes and press "Scan".', int(m.sqrt(display_area*0.0017)),
             c.black, display_width, display_height//4, 'center of box')
        # (above) display instruction for the user

        # (below) Setting the coordinate for the picture of the hand to guide the user
        hand_w, hand_h = pic.hand.get_rect().size
        hand_x = display_width // 2 + display_width // 24 + \
                 (display_width // 2 - display_width // 24 - hand_w)// 2
        hand_y = display_height // 4 + (display_height // 2 - hand_h)//2

        gameDisplay.blit(pic.hand, (hand_x, hand_y))
        # Display the hand that guides the user on what to do

        pressed = button('center', display_width//2, display_height // 9 * 8, button_w,
                         button_h, c.red_dim, c.red_lit, 'Scan', button_font_size, c.black,
                         'scan_screen', 'hand_scan')
        # (above) create button to scan the hand and get the state of whether
        # this button was pressed

        if pressed: # if the scan button was pressed,
            hand_hist = hand_scan(frame) # get the hand histogram
            cap.release() # release the camera from taking video
            scanned = 1 # indicate that the hand is scanned by storing value in the var.
            game_screen() # go into the Game screen

        ut.update_screen(clock) # Update screen
        ut.quit_x() # Close window if 'X' button (top right) or ESC key is pressed
    cap.release() # release the camera from capturing video
```

- **Display**
    a. frame_adjust(frame, hori_1, hori_2, vert_1, vert_2, scan_now=False)

```
# Function to put in video in the Scan or Game screen
def frame_adjust(frame, hori_1, hori_2, vert_1, vert_2, scan_now=False):
    '''Put the video in place in the Scan or Game screen with green or
    reddish pink rectangles respectively'''
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    # Convert video from BGR to RGB
    y, x, _ = frame.shape
    # Get height and width of video
    crop_x1, crop_x2, crop_y1, crop_y2 = int(x*0.01563), int(x*0.33281),\
                                         int(y*0.26042), int(y*0.73958)
    # Setting coordinates for video

    if scan_now:
        # if current screen is Scan screen,
        frame = translucent_obj(frame, draw_rect_right, 1.0)
        # Draw green rectangle in the video
    else:
        # else, in Game Screen,
        cv2.rectangle(frame, (crop_x1, crop_y2), (crop_x2, crop_y1), c.green_lit, 5)
        # Draw reddish pink rectangles in the video

    frame = np.rot90(frame) # Rotating the frame

    frame = pygame.surfarray.make_surface(frame)
    # copy video frame's array to a new surface
    frame = pygame.transform.scale(frame, (hori_2-hori_1, vert_2-vert_1))
    # resize the frame to fit the game screen
    gameDisplay.blit(frame, (hori_1, vert_1))
    # Display the video in the game screen
```

- **Hand histogram**
    a. draw_rect_right()

```
# Function to draw reddish pink rectangles to scan hand
def draw_rect_right(frame, return_list_only=False, color=(200, 0, 255), thickness=2):
    '''Draw reddish pink rectangles in the video and return list of the coordinates
    of the rectangles' starting and ending point or only return the list only if
    return_list_only=True'''
    y, x, _ = frame.shape
    # getting the dimension of the video
    num_box = 12
    # specifying the number of rectangles to be drawn
    start_rect = []
    # initializing the var. to store the starting coordinate of each rectangles
    start = [x * 0.10, y * 0.35]
    # Setting the starting point of the first rectangles
    end = [x * 0.23, y * 0.65]
    # Setting the ending point of the last rectangles

    spacing = [((end[0] - start[0]) - (3 * 10)) // 2, ((end[1] - start[1]) - (4 * 10)) //
    # Setting the space between the rectangles

    count_x = 0
    # Count variable to assist in calculating the rectangles x position
    count_y = 0
    # Count variable to assist in calculating the rectangles y position

    for i in range(1, num_box + 1):
        # loop to store variable of starting point of each rectangles
        start_rect.append([int(start[0] + count_x * (10 + spacing[0])),
                           int(start[1] + count_y * (10 + spacing[1]))])
        # (above) appending x and y coordinate of the starting point of the rectangles
        count_x = count_x + 1
        # increment count to store the x coordinate of the next rectangle
        if i % 3 == 0:
            # if 3, 6, 9, 12 boxes has been created,
            count_x = 0
            # reset count
            count_y = count_y + 1
            # increment count to store the y coordinate of the next row of rectangle

    start_rect = np.array(start_rect)
    # convert list to numpy array to do calculation
    end_rect = start_rect + 10
    # creating the list that holds the coordinates of the ending point of the rectangles

    if not return_list_only:
        # if going to draw rectangles
        for i in range(0, num_box):
            # loop to draw rectangles
            frame = cv2.rectangle(frame, (start_rect[i][0], start_rect[i][1]),
                                  (end_rect[i][0], end_rect[i][1]), color, thickness)
        return frame, start_rect, end_rect
        # return frame with drawn rectangles and list of the coordinates of the rectangles
    else:
        # if only the list of the coordinates of rectangles is needed,
        return start_rect, end_rect
        # return only the list of coordinates of the rectangles
```

b. hand_scan(frame)

```
# Function to get hand histogram
def hand_scan(frame):
    '''Return hand histogram'''
    start_rect, end_rect = draw_rect_right(frame, True)
    # get the list of coordinates of the rectangles
    hand_hist = set_hand_hist(frame, start_rect, end_rect)
    # get the hand histogram
    return hand_hist
    # return hand histogram
```

c. set_hand_hist()

```
# Function to calculate and get the hand histogram
def set_hand_hist(frame, start_rect, end_rect):
    '''Get hand histogram from the rectangles(more like squares though)'''
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # convert frame of the video to HSV colorspace
    roi = np.zeros([120, 10, 3], dtype=hsv.dtype)
    # create a numpy array of zeros to store the value of histogram

    size = int(start_rect.size / 2)
    # get the number of rectangles

    for i in range(size):
        # loop to store values of histogram in the rectangles to
        # the roi numpy array
        roi[i * 10:i * 10 + 10, 0:10] = hsv[start_rect[i][1]:end_rect[i][1],
                                            start_rect[i][0]:end_rect[i][0]]
        # (above) replacing the values of roi numpy array to the histogram values

    hand_hist = cv2.calcHist([roi], [0, 1], None, [180, 256], [0, 180, 0, 256])
    # calculate and store the calculated histogram
    cv2.normalize(hand_hist, hand_hist, 0, 255, cv2.NORM_MINMAX)
    # normalize the histogram
    return hand_hist
    # return the hand histogram
```

d. translucent_obj()

```
# Function to draw reddish pink rectangles on the copy of the frame and
# combine the copy and the frame
def translucent_obj(frame, action=None, alpha=0.5):
    '''Put a translucent object in the frame'''
    img = frame.copy()
    # create a copy of the frame
    action(img)
    # do the action to the copy
    frame = cv2.addWeighted(img, alpha, frame, 1 - alpha, 0)
    # combine the frame and the copy
    return frame
    # return the combined frame
```

Game screen

- **Main**
    a. game_screen()

```
# Function to create Game screen
def game_screen():
    ''' create game screen'''
    music.stop()
    # stop playing the music played previously
    music.game()
    # start playing the music for game screen
    global score_me
    # get the score for user
    global score_com
    # get the score for computer
    global cap
    # get the video capture
    global matches
    # get the number of matches played
    if matches == 0:
        # if no matches are played yet,
        score_me = score_com = 0
        # set both scores to zero
    cap = cv2.VideoCapture(0)
    # start taking video
    start = time.time()
    # start timing to change the color of the circles and the words that helps
    # the user to gauge the timing
    while cap.isOpened():
        # while the camera is really taking the video,
        _, frame = cap.read()
        # get the frame from the camera

        hori_1, hori_2, vert_1, vert_2 = init_game_screen(True)
        # (above) create the game screen background and get the values to adjust
        # the frame into game screen
        frame_adjust(frame, hori_1, hori_2, vert_1, vert_2)
        # adjust the frame into the game screen
        you_com()
        # display the 'You' and 'Com' in the screen

        detect_now = SPS_timing(time.time()-start)
        # display the circles and words that helps the user to gauge the timing
        # (above) also get a value to tell if it is time for the computer to detect
        # the gesture

        if detect_now:
            # if it is time to detect,
            gesture = detect(frame)
            # get the computer to predict the user's gesture
            rand_gesture = random_gesture()
            # get the gesture for the computer
            display_gesture(gesture, rand_gesture, frame)
            # display game result screen for 2 seconds
            # (above) also calculate the scores for both sides and the number of matches done
            start = time.time()
            # (above) reset the starting time to change the color of the circles and
            # the words that helps the user to gauge the timing again
```

```
        if matches == 5:
            # if the number of matches done is 5,
            cap.release()
            # release the camera from taking frames
            matches = 0
            # reset the number of matches done
            result_screen(score_me, score_com)
            # display the result screen

        ut.update_screen(clock)
        # Update screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed

    cap.release()
    # release the camera from taking frames
```

- **Computer Gesture**
    a. random_gesture()

```
# Function to randomly choose a gesture ('Scissor', 'Paper', 'Stone') for the computer
def random_gesture():
    ''' get a gesture for the computer by choosing gesture randomly'''
    gesture_int = random.randint(1, 3)
    # choose an integer (1, 2, 3)
    # (below) returns gesture based on the randomly chosen integer
    if gesture_int == 1:
        return 'Scissor'
    if gesture_int == 2:
        return 'Paper'
    if gesture_int == 3:
        return 'Stone'
```

- **Display**
    a. init_game_screen(run=True)

```
# Function to display the Screen/Game background
def init_game_screen(run=True):
    '''create scan/game screen background and display it or
     return the starting and ending coordinates to position the video
     and resize the video to an appropriate size'''
    vert_bor_x, vert_bor_y, vert_bor_w, vert_bor_h =\
        display_width // 2 - display_width // 24, 0, display_width // 12, display_height
    # (above) Setting coordinates and size of vertical border in the screen
    horz_top_x, horz_top_y, horz_top_w, horz_top_h = 0, 0, display_width, display_height // 4
    # (above) Setting coordinates and size of top horizontal bar in the screen
    horz_bottom_x, horz_bottom_y, horz_bottom_w, horz_bottom_h =\
        0, display_height // 9 * 7, display_width, display_height // 4
    # (above) Setting coordinates and size of bottom horizontal border in the screen
    if run:
        # if the scan/game screen needs to be displayed,
        gameDisplay.fill(c.yellow_green)
        # fill the screen with yellow green color
        pygame.draw.rect(gameDisplay, c.grey_green, (vert_bor_x, vert_bor_y, vert_bor_w,
                                                     vert_bor_h))
        # draw vertical border
        pygame.draw.rect(gameDisplay, c.light_green_blue, (horz_top_x, horz_top_y, horz_top_w,
                                                           horz_top_h))
```

```
        # draw top horizontal bar
        pygame.draw.rect(gameDisplay, c.grey_yellow, (horz_bottom_x, horz_bottom_y, horz_bottom_w,
                                                      horz_bottom_h))
        # draw bottom horizontal bar
        pause_button()
        # create pause button
    hori_1, hori_2, vert_1, vert_2 =\
        offset_x, vert_bor_x - offset_x, horz_top_h + offset_y, horz_bottom_y - offset_y
    # (above) get the starting and ending coordinates to position the video and
    # resize the video to an appropriate size
    return hori_1, hori_2, vert_1, vert_2
    # return starting and ending coordinates (of the video)
```

b. SPS_timing(time)

```
# Function to display words ('Ready', 'Scissor', 'Paper', 'Stone'), circles with text
# and changing colors and sound for timing the move
def SPS_timing(time):
    '''Display words ('Ready', 'Scissor', 'Paper', 'Stone'), circles with text and
    changing colors and sound for timing the move
    also returns True value when gesture needs to be detected'''

    # (below) Setting big and small circles' coordinates and radius
    b_circle1_x, b_circle1_y = display_width // 3, int(display_height * (3 / 16))
    b_circle2_x, b_circle2_y = display_width // 2, int(display_height * (3 / 16))
    b_circle3_x, b_circle3_y = int(display_width * (2 / 3)), int(display_height * (3 / 16))
    s_circle1_x, s_circle1_y = int(display_width * (5 / 12)), int(display_height * (3 / 16))
    s_circle2_x, s_circle2_y = int(display_width * (7 / 12)), int(display_height * (3 / 16))
    b_circle_rad = int(m.sqrt(display_area*0.00533/PI))
    s_circle_rad = int(m.sqrt(display_area*0.00192/PI))

    timing_text = int(m.sqrt(display_area*0.00678))
    # Setting font size of text in the big circles
    lit = c.orange
    # Setting color of circles when lit
    dim = c.pink_red
    # Setting color of circle when not lit
    ready_time = 2
    # Setting duration to display text 'Ready'
    detect_now = False
    # Initializing var.
    global init_sound
    # Using global variable init_sound
    # (below) Displaying words and circles within a certain duration to gauge the move
    if time < ready_time:
        text('Ready?', timing_text, c.black, display_width, display_height // 8, 'center of box')
        pygame.draw.circle(gameDisplay, dim, (b_circle1_x, b_circle1_y), b_circle_rad)
        pygame.draw.circle(gameDisplay, dim, (s_circle1_x, s_circle1_y), s_circle_rad)
        pygame.draw.circle(gameDisplay, dim, (b_circle2_x, b_circle2_y), b_circle_rad)
        pygame.draw.circle(gameDisplay, dim, (s_circle2_x, s_circle2_y), s_circle_rad)
        pygame.draw.circle(gameDisplay, dim, (b_circle3_x, b_circle3_y), b_circle_rad)
        text('3', b_circle_rad*2, c.black, b_circle1_x, b_circle1_y+5, 'center point')
        text('2', b_circle_rad*2, c.black, b_circle2_x, b_circle2_y + 5, 'center point')
        text('1', b_circle_rad*2, c.black, b_circle3_x, b_circle3_y + 5, 'center point')
```

```
elif ready_time < time < ready_time + 0.5:
    init_sound +=1
    text('Scissor...', timing_text, c.black, display_width, display_height // 8, 'center of box')
    pygame.draw.circle(gameDisplay, lit, (b_circle1_x, b_circle1_y), b_circle_rad)
    pygame.draw.circle(gameDisplay, dim, (s_circle1_x, s_circle1_y), s_circle_rad)
    pygame.draw.circle(gameDisplay, dim, (b_circle2_x, b_circle2_y), b_circle_rad)
    pygame.draw.circle(gameDisplay, dim, (s_circle2_x, s_circle2_y), s_circle_rad)
    pygame.draw.circle(gameDisplay, dim, (b_circle3_x, b_circle3_y), b_circle_rad)
    text('3', b_circle_rad*2, c.black, b_circle1_x, b_circle1_y+5, 'center point')
    text('2', b_circle_rad*2, c.black, b_circle2_x, b_circle2_y + 5, 'center point')
    text('1', b_circle_rad*2, c.black, b_circle3_x, b_circle3_y + 5, 'center point')
    gameDisplay.blit(pic.scissors, (gesture_pic_x, gesture_pic_y))
    if init_sound == 1:
        sound.timing()

elif ready_time + 0.5 < time < ready_time + 0.5*2:
    init_sound = 0
    text('Scissor...', timing_text, c.black, display_width, display_height // 8, 'center of box')
    pygame.draw.circle(gameDisplay, lit, (b_circle1_x, b_circle1_y), b_circle_rad)
    pygame.draw.circle(gameDisplay, lit, (s_circle1_x, s_circle1_y), s_circle_rad)
    pygame.draw.circle(gameDisplay, dim, (b_circle2_x, b_circle2_y), b_circle_rad)
    pygame.draw.circle(gameDisplay, dim, (s_circle2_x, s_circle2_y), s_circle_rad)
    pygame.draw.circle(gameDisplay, dim, (b_circle3_x, b_circle3_y), b_circle_rad)
    text('3', b_circle_rad*2, c.black, b_circle1_x, b_circle1_y+5, 'center point')
    text('2', b_circle_rad*2, c.black, b_circle2_x, b_circle2_y + 5, 'center point')
    text('1', b_circle_rad*2, c.black, b_circle3_x, b_circle3_y + 5, 'center point')
    gameDisplay.blit(pic.scissors, (gesture_pic_x, gesture_pic_y))

elif ready_time + 0.5*2 < time < ready_time + 0.5*3:
    init_sound +=1
    text('Paper...', timing_text, c.black, display_width, display_height // 8, 'center of box')
    pygame.draw.circle(gameDisplay, lit, (b_circle1_x, b_circle1_y), b_circle_rad)
    pygame.draw.circle(gameDisplay, lit, (s_circle1_x, s_circle1_y), s_circle_rad)
    pygame.draw.circle(gameDisplay, lit, (b_circle2_x, b_circle2_y), b_circle_rad)
    pygame.draw.circle(gameDisplay, dim, (s_circle2_x, s_circle2_y), s_circle_rad)
    pygame.draw.circle(gameDisplay, dim, (b_circle3_x, b_circle3_y), b_circle_rad)
    text('3', b_circle_rad*2, c.black, b_circle1_x, b_circle1_y+5, 'center point')
    text('2', b_circle_rad*2, c.black, b_circle2_x, b_circle2_y + 5, 'center point')
    text('1', b_circle_rad*2, c.black, b_circle3_x, b_circle3_y + 5, 'center point')
    gameDisplay.blit(pic.paper, (gesture_pic_x, gesture_pic_y))
    if init_sound == 1:
        sound.timing()


elif ready_time + 0.5*3 < time < ready_time + 0.5*4:
    init_sound = 0
    text('Paper...', timing_text, c.black, display_width, display_height // 8, 'center of box')
    pygame.draw.circle(gameDisplay, lit, (b_circle1_x, b_circle1_y), b_circle_rad)
    pygame.draw.circle(gameDisplay, lit, (s_circle1_x, s_circle1_y), s_circle_rad)
    pygame.draw.circle(gameDisplay, lit, (b_circle2_x, b_circle2_y), b_circle_rad)
    pygame.draw.circle(gameDisplay, lit, (s_circle2_x, s_circle2_y), s_circle_rad)
    pygame.draw.circle(gameDisplay, dim, (b_circle3_x, b_circle3_y), b_circle_rad)
    text('3', b_circle_rad*2, c.black, b_circle1_x, b_circle1_y+5, 'center point')
    text('2', b_circle_rad*2, c.black, b_circle2_x, b_circle2_y + 5, 'center point')
    text('1', b_circle_rad*2, c.black, b_circle3_x, b_circle3_y + 5, 'center point')
    gameDisplay.blit(pic.paper, (gesture_pic_x, gesture_pic_y))
```

```
    elif ready_time + 0.5*3 < time < ready_time + 0.5*4:
        init_sound = 0
        text('Paper...', timing_text, c.black, display_width, display_height // 8, 'center of box')
        pygame.draw.circle(gameDisplay, lit, (b_circle1_x, b_circle1_y), b_circle_rad)
        pygame.draw.circle(gameDisplay, lit, (s_circle1_x, s_circle1_y), s_circle_rad)
        pygame.draw.circle(gameDisplay, lit, (b_circle2_x, b_circle2_y), b_circle_rad)
        pygame.draw.circle(gameDisplay, lit, (s_circle2_x, s_circle2_y), s_circle_rad)
        pygame.draw.circle(gameDisplay, dim, (b_circle3_x, b_circle3_y), b_circle_rad)
        text('3', b_circle_rad*2, c.black, b_circle1_x, b_circle1_y+5, 'center point')
        text('2', b_circle_rad*2, c.black, b_circle2_x, b_circle2_y + 5, 'center point')
        text('1', b_circle_rad*2, c.black, b_circle3_x, b_circle3_y + 5, 'center point')
        gameDisplay.blit(pic.paper, (gesture_pic_x, gesture_pic_y))


    elif time > ready_time + 0.5*4:
        text('Stone!', timing_text, c.black, display_width, display_height // 8, 'center of box')
        pygame.draw.circle(gameDisplay, lit, (b_circle1_x, b_circle1_y), b_circle_rad)
        pygame.draw.circle(gameDisplay, lit, (s_circle1_x, s_circle1_y), s_circle_rad)
        pygame.draw.circle(gameDisplay, lit, (b_circle2_x, b_circle2_y), b_circle_rad)
        pygame.draw.circle(gameDisplay, lit, (s_circle2_x, s_circle2_y), s_circle_rad)
        pygame.draw.circle(gameDisplay, lit, (b_circle3_x, b_circle3_y), b_circle_rad)
        text('3', b_circle_rad*2, c.black, b_circle1_x, b_circle1_y+5, 'center point')
        text('2', b_circle_rad*2, c.black, b_circle2_x, b_circle2_y + 5, 'center point')
        text('1', b_circle_rad*2, c.black, b_circle3_x, b_circle3_y + 5, 'center point')
        detect_now = True
        # Setting var. to indicate program to recognize gesture now

    return detect_now
    # Return var. to indicate whether to recognize gesture or not
```

c. you_com()

```
# Function to display text ('You' and 'Com') in the scan/game screen
def you_com():
    '''Display 'You' and 'Com' text in the scan/game screen'''
    you_w, you_h = text_size('You', you_com_text)
    # get width and height of 'You' text
    text('You', you_com_text, c.red_dim, int(display_width//2-display_width//24-you_w),
        int(display_height//9*7-you_h), '', c.white)
    # (above) creating and displaying text 'You'
    com_w, com_h = text_size('Com', you_com_text)
    # get width and height of 'Com' text
    text('Com', you_com_text, c.blue_dim, int(display_width-com_w),
        int(display_height//9*7-com_h), '', c.white)
    # (above) creating and displaying text 'Com'
```

d. win_lose(gesture, rand_gesture)

```
# Function to get the result of the match for both user and computer
def win_lose(gesture, rand_gesture):
    # (below) initializing the var.s to store the result of the match
    # for both user and computer
    result_me = ''
    result_com = ''
    # (below) storing the result for both user and computer
    if gesture == rand_gesture:
        result_me = result_com = 'Draw'
```

```
    elif gesture == 'Scissor' and rand_gesture == 'Paper':
        result_me = 'Win'
        result_com = 'Lose'

    elif gesture == 'Stone' and rand_gesture == 'Paper':
        result_me = 'Lose'
        result_com = 'Win'

    elif gesture == 'Paper' and rand_gesture == 'Scissor':
        result_me = 'Lose'
        result_com = 'Win'

    elif gesture == 'Stone' and rand_gesture == 'Scissor':
        result_me = 'Win'
        result_com = 'Lose'

    elif gesture == 'Paper' and rand_gesture == 'Stone':
        result_me = 'Win'
        result_com = 'Lose'

    elif gesture == 'Scissor' and rand_gesture == 'Stone':
        result_me = 'Lose'
        result_com = 'Win'
    elif gesture == 'Nothing':
        result_me = 'Lose'
        result_com = 'Win'
    elif gesture == '???':
        result_me = 'Lose'
        result_com = 'Win'

    return result_me, result_com
    # returning the result of both user and computer
```

e. display_gesture(gesture, rand_gesture, frame)

```
# Function to display game result screen for 2 seconds
def display_gesture(gesture, rand_gesture, frame):
    '''display the gesture of user and computer (for 2 seconds),
    sound the sound for 'Win', 'Lose' and 'Draw',
    increment the score when necessary for both user and computer,
    increment the number of matches done'''
    global matches
    # get the number of matches played
    matches += 1
    # increment the number of matches
    global score_me
    # get the score for user
    global score_com
    # get the score for computer
    start = time.time()
    # start timing the duration to display gesture
    result_me, result_com = win_lose(gesture, rand_gesture)
    # get the result of the match for both sides
```

```
if result_me == 'Win':
    # if user win,
    score_me += 1
    # increment score for user
    sound.win()
    # sound the sound when user wins
elif result_me == 'Draw':
    # if the match is a draw,
    # (below) increment scores for both sides
    score_me += 1
    score_com += 1
    sound.draw()
    # sound the sound when draw
elif result_me == 'Lose':
    # if user lose,
    score_com += 1
    # increment score for computer
    sound.lose()
    # sound the sound when user lose
while True:
    # infinite loop
    hori_1, hori_2, vert_1, vert_2 = init_game_screen(False)
    # get values to adjust the frame in the game result screen
    frame_adjust(frame, hori_1, hori_2, vert_1, vert_2)
    # adjust the frame in the game result screen
    text(gesture, result_gesture_text, c.black, int(display_width*0.01953),
         int(display_height*0.80208))
    # (above) display text to write user's gesture that the computer has predicted
    rand_gesture_w, _ = text_size(rand_gesture, result_gesture_text)
    # get the width of computer gesture's word
    text(rand_gesture, result_gesture_text, c.black,
         int(display_width-display_width*0.01953-rand_gesture_w),
         int(display_height*0.80208))
    # (above) display text to write computer's gesture
    if rand_gesture == 'Scissor':
        # if computer's gesture is 'Scissor',
        gameDisplay.blit(pic.scissors, (gesture_pic_x, gesture_pic_y))
        # display picture of scissor
    elif rand_gesture == 'Paper':
        # if computer's gesture is 'Paper',
        gameDisplay.blit(pic.paper, (gesture_pic_x, gesture_pic_y))
        # display picture of paper
    elif rand_gesture == 'Stone':
        # if computer's gesture is 'Stone',
        gameDisplay.blit(pic.stone, (gesture_pic_x, gesture_pic_y))
        # display picture of stone

    text(result_me, result_gesture_text, c.black, int(display_width*0.01953),
         int(display_height*0.03125))
    # (above) display text to write result for user
    result_com_w, _ = text_size(result_com, result_gesture_text)
    # get the width of computer's result's word
    text(result_com, result_gesture_text, c.black,
         display_width-int(display_width*0.01953)-result_com_w,
         int(display height*0.03125))
```

- **Processing frames**

a. clearer(frame, kernel_size=5, iteration=1)

```
# Function to make binary image clearer
def clearer(frame, kernel_size=5, iteration=1):
    '''to reduce noise of the image and make it clear and distinct'''
    kernel = np.ones((kernel_size,kernel_size),np.uint8)
    # creating kernel of filter
    frame = cv2.erode(frame,kernel,iterations = iteration)
    # eroding the bonder of the white in the image
    frame = cv2.morphologyEx(frame, cv2.MORPH_OPEN, kernel)
    # remove noise
    frame = cv2.morphologyEx(frame, cv2.MORPH_CLOSE, kernel)
    # close up small holes

    return frame
    # return filtered frame
```

b. crop_binary(frame)

```
# Function to prepare images to feed into the CNN for prediction
def crop_binary(frame):
    '''crop the frame where the green rectangle are and convert the cropped area of
    the frame to binary
    also process the binary frame to reduce noise and returns that binary frame'''
    y, x, _ = frame.shape
    # get the dimension of the frame
    crop_x1, crop_x2, crop_y1, crop_y2 = int(x*0.01563), int(x*0.33281),\
                                         int(y*0.26042), int(y*0.73958)
    # (above) calculate and store the coordinate of the frame to be cropped
    crop = frame[crop_y1:crop_y2, crop_x1:crop_x2]
    # crop the frame and storethe value into a var.
    thresh = hist_mask(crop, hand_hist)
    # convert the cropped frame to binary
    clear = clearer(thresh, 5, 1)
    # process the binary frame such that it has lesser noise
    return clear
    # return the clear binary cropped frame
```

c. hist_mask(frame, hand_hist)

```
# Function to make the cropped frame into binary image
def hist_mask(frame, hand_hist):
    '''Return binary image of the skin where only the skin is white and
    everything else is black'''
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # convert the frame to HSV colorspace
    dst = cv2.calcBackProject([hsv], [0, 1], hand_hist, [0, 181, 0, 256], 1)
    # get a black and white image
    # (above) intensity of white depends on how well the pixel value in the image fits
    # (above) the more it fits, the whiter the pixel
    disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
    # create a kernel to filter the noise of the image
```

```
    cv2.filter2D(dst, -1, disc, dst)
    # filter the noise of the image
    _, thresh1 = cv2.threshold(dst, 0, 255, 0)
    # get the binary image of the filtered black and white image
    return thresh1
    # return the binary image
```

- **Prediction**

    a. detect(frame)

```
# Function to predict gesture when necessary
def detect(frame):
    '''Returns predicted gesture by the computer'''
    clear = crop_binary(frame)
    # get a cropped binary clear image
    white_ratio, h_clear = nonzero_ratio(clear)
    # get non-zero percentage and resized binary image

    if white_ratio > 14:
        # if there is more than 14% of non-zero,
        gesture = predict(h_clear)
        # predict and get gesture of the user
    else:
        # if there is less than 14% of non-zero,
        gesture = 'Nothing'
        # assume there is no gesture given by the user
    return gesture
    # return gesture predicted/assumed
```

    b. nonzero_ratio(clear)

```
# Function to get the percentage of non-zero in the cropped binary image and
# get the cropped binary image resized to 100 x 100
def nonzero_ratio(clear):
    '''Return the percentage of non-zero in the binary image and
    return binary image resized to 100 x 100'''
    h_clear = cv2.resize(clear, (100, 100)) # resize image to 100 x 100
    nonzero = cv2.countNonZero(h_clear) # get the number of non-zero pixel
    nonzero_ratio = (nonzero / (100 * 100)) * 100
    # get the percentage of non-zero pixel in the binary image
    return nonzero_ratio, h_clear
    # return the percentage of non-zero pixel and the binary image resized to 100 x 100
```

    c. predict(h_clear)

```
# Function to predict the gesture in the image using CNN
def predict(h_clear):
    '''Return gesture predicted by the model or '???' when the confidence is low'''
    np_clear = h_clear.reshape(-1, 100, 100, 1)
    # reshape the array of image to 100 x 100 with one color channel
    np_clear = np.array(np_clear)
    # convert the reshaped array to numpy array
    np_clear = np_clear / 255
    # divide the whole array by 255
```

```
    prediction = model.predict([np_clear])
    # get the array of confidence of all classes from CNN
    predictions = np.where(prediction == np.amax(prediction))
    # get the index of the highest confidence in the array
    if predictions[1] == 0 and (prediction[0][0] >= 0.9):
        # if the index is 0 and the confidence level is more or equal to 0.9,
        return 'Stone'
    elif predictions[1] == 1 and (prediction[0][1] >= 0.9):
        # if the index is 1 and the confidence level is more or equal to 0.9,
        return 'Paper'
    elif predictions[1] == 2 and (prediction[0][2] >= 0.9):
        # if the index is 2 and the confidence level is more or equal to 0.9,
        return 'Scissor'
    else:
        # if none of them has a confidence level of more or equal to 0.9,
        return '???'
```

Result Screen

- **Main**
    a. result_screen(score_me, score_com)

```
# Function to show result screen
def result_screen(score_me, score_com):
    ''' create result screen'''
    if score_me == score_com:
        # if both scores are equal,
        music.stop()
        # stop playing the music previously played
        music.draw()
        # start playing music when the matches is a draw
    elif score_me > score_com:
        # if user's score is more than computer's score,
        music.stop()
        # stop playing the music previously played
        music.win()
        # start playing music when user win
    elif score_com > score_me:
        # if computer's score is more than user's score,
        music.stop()
        # stop playing the music previously played
        music.lose()
        # start playing music when user lose
    while True:
        # infinite loop
        gameDisplay.blit(pic.background, (0, 0))
        # display default background
```

```
    # (below) Calculations to get values to adjust the coordinate and
    # the font size of the words to be displayed in the result screen
    result_msg_size = int(m.sqrt(display_area*0.01846))
    score_header_size = int(m.sqrt(display_area*0.0017))
    score_text_size = int(m.sqrt(display_area*0.00678))
    score_header_pos = int(display_height*0.41146)
    score_text_pos = int(display_height*0.52083)
    note_text_size = int(m.sqrt(display_area*0.00061))
    note_1_pos = int(display_height*0.92188)
    note_2_pos = int(display_height*0.95313)

    if score_me == score_com:
        # if both scores are equal,
        # (below) display texts when user and computer draws
        text('Draw? No fun here.', result_msg_size, c.black, display_width//2,
             display_height//4, 'center point', c.grey_yellow)
        text('Your Score:', score_header_size, c.black, display_width//3,
             score_header_pos, 'center point', c.yellow_green)
        text(str(score_me), score_text_size, c.white, display_width//3,
             score_text_pos, 'center point', None)
        text('Computer Score:', score_header_size, c.black, display_width // 3*2,
             score_header_pos, 'center point', c.light_green_blue)
        text(str(score_com), score_text_size, c.white, display_width // 3*2,
             score_text_pos, 'center point', None)

    elif score_me > score_com:
        # if user's score is more than computer's score,
        # (below) display texts when user win
        text('Tsk. You Win.', result_msg_size, c.black, display_width//2,
             display_height//4, 'center point', c.grey_green)
        text('Your Score:', score_header_size, c.black, display_width//3,
             score_header_pos, 'center point', c.yellow_green)
        text(str(score_me), score_text_size, c.white, display_width//3, score_text_pos,
             'center point', None)
        text('Computer Score:', score_header_size, c.black, display_width // 3*2,
             score_header_pos, 'center point', c.light_green_blue)
        text(str(score_com), score_text_size, c.white, display_width // 3*2,
             score_text_pos, 'center point', None)

    elif score_com > score_me:
        # if computer's score is more than user's score,
        # (below) display texts when user lose
        text('Ha-Hah! You Lose!', result_msg_size, c.black, display_width//2,
             display_height//4, 'center point', c.pink_red)
        text('Your Score:', score_header_size, c.black, display_width//3,
             score_header_pos, 'center point', c.yellow_green)
        text(str(score_me), score_text_size, c.white, display_width//3, score_text_pos,
             'center point', None)
        text('Computer Score:', score_header_size, c.black, display_width // 3*2,
             score_header_pos, 'center point', c.light_green_blue)
        text(str(score_com), score_text_size, c.white, display_width // 3*2, score_text_pos,
             'center point', None)

    # (below) create buttons ('Main Menu', 'Scan again', 'Play again', 'Instruction', 'Quit')
    button('center', display_width//4, display_height//3*2, button_w, button_h,
           c.orange_dim, c.orange_lit, 'Main Menu', button_font_size, c.black,
           'result', 'intro')
    button('center', display_width//4 * 2, display_height//3*2, button_w, button_h,
           c.purple_dim, c.purple_lit, 'Scan again', button_font_size, c.black,
           'result', 'scan_screen')
```

```
        button('center', display_width//4 * 3, display_height//3*2, button_w, button_h,
               c.green_dim, c.green_lit, 'Play again', button_font_size, c.black,
               'result', 'game')
        button('center', display_width //3, display_height //6*5, button_w, button_h,
               c.yellow_dim, c.yellow_lit, 'Instruction', button_font_size, c.black,
               'result', 'tutt_1')
        button('center', display_width//3*2, display_height//6*5, button_w, button_h,
               c.red_dim, c.red_lit, 'Quit', button_font_size, c.black, 'result', 'quit')

        # (below) display texts for the user to decide whether to scan again before playing again
        text('If lighting have changed or the next player is a different person,',
             note_text_size, c.black, display_width//2, note_1_pos, 'center point', c.white)
        text("please select 'Scan again' if you want to play again.", note_text_size,
             c.black, display_width//2, note_2_pos, 'center point', c.white)

        ut.update_screen(clock)
        # Update screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

Pause Screen

- **Main**

  a. pause_menu()

```
# Function to create Pause menu
def pause_menu():
    '''Create Pause menu'''
    while True:
        gameDisplay.blit(pic.background, (0, 0))
        # Display default background
        text('Paused', paused_text_size, c.black, display_width, display_height//2,
             'center of box')
        # Display pause menu title, 'Paused'
        # Creating buttons to go into Main menu, Scan, Game, Tutorial 1 screens and
        # Quit button to close window
        button('center', display_width//4, display_height//3*2, button_w, button_h,
               c.orange_dim, c.orange_lit, 'Main Menu', button_font_size, c.black,
               'pause', 'intro')
        button('center', display_width//4 * 2, display_height//3*2, button_w, button_h,
               c.purple_dim, c.purple_lit, 'Scan', button_font_size, c.black, 'pause',
               'scan_screen')
        button('center', display_width//4 * 3, display_height//3*2, button_w, button_h,
               c.green_dim, c.green_lit, 'Play', button_font_size, c.black, 'pause', 'game')
        button('center', display_width //3, display_height //6*5, button_w, button_h,
               c.yellow_dim, c.yellow_lit, 'Instruction', button_font_size, c.black,
               'pause', 'tutt_1')
        button('center', display_width//3*2, display_height//6*5, button_w, button_h,
               c.red_dim, c.red_lit, 'Quit', button_font_size, c.black, 'pause', 'quit')
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

- **Display**
  a. pause_scan_first()

```
# Function to create Pause menu with message to tell the user to scan the hand first
def pause_scan_first():
    '''Create Pause menu with message scan first'''
    sound.scan_first()
    # sound the Scan first message sound
    pygame.mixer.music.set_volume(0.3)
    # Decrease the volume of music that is previously played
    scan_first_text_size = int(m.sqrt(display_area*0.00678))
    # Setting the message font size
    start = time.time() # start timing the duration of the message
    while True:
        gameDisplay.blit(pic.background, (0, 0))
        # Display the default background
        text('Paused', paused_text_size, c.black, display_width, display_height//2,
             'center of box')
        # Display the title of Pause menu
        text('Please scan your hand first.', scan_first_text_size, c.white, display_width//2,
             display_height // 2, 'center point')
        # Display message to scan first
        # (below) Creating buttons to go into Main menu, Scan, Game, Tutorial 1 screens and
        # Quit button to close window
        button('center', display_width//4, display_height//3*2, button_w, button_h,
               c.orange_dim, c.orange_lit, 'Main Menu', button_font_size, c.black, 'pause',
               'intro')
        button('center', display_width//4 * 2, display_height//3*2, button_w, button_h,
               c.purple_dim, c.purple_lit, 'Scan', button_font_size, c.black, 'pause',
               'scan_screen')
        button('center', display_width//4 * 3, display_height//3*2, button_w, button_h,
               c.green_dim, c.green_lit, 'Play', button_font_size, c.black, 'pause', 'game')
        button('center', display_width//3, display_height//6*5, button_w, button_h, c.yellow_dim,
               c.yellow_lit, 'Instruction', button_font_size, c.black, 'pause', 'tutt_1')
        button('center', display_width//3*2, display_height//6*5, button_w, button_h, c.red_dim,
               c.red_lit, 'Quit', button_font_size, c.black, 'pause', 'quit')
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
        if time.time()-start > 3:
            # if duration of message displayed is more than 3 seconds,
            pause_menu()
            # go into the normal Pause menu
```

- **Pause Button**
  a. pause_button()

```
# Function to create pause button
def pause_button():
    button('top right', display_width, 0, pause_button_size, pause_button_size, c.red_dim,
           c.red_lit, '', '', '', 'game', 'paused')
    # (above) creating button to go into Pause screen
    # (below) creating black rectangles in the pause button to indicate that
    # it is the pause button
    pygame.draw.rect(gameDisplay, c.black, (black_bar1_x, black_bar1_y, black_bar1_w,
                                            black_bar1_h))
    pygame.draw.rect(gameDisplay, c.black, (black_bar2_x, black_bar2_y, black_bar2_w,
                                            black_bar2_h))
```

Tutorial (Instructions)

- **Main**

a. tutorial_one()

```
# Functions to create Instruction screens
def tutorial_one():
    '''Go to the Tutorial 1 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_1, (0, 0))
        # display image of Tutorial screen 1
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h,
               c.red_dim, c.red_lit, 'Main Menu', button_font_size, c.black,
               'tutt_1', 'intro')
        # (above) creating button to go into Main menu
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h,
               c.red_dim, c.red_lit, 'Next', button_font_size, c.black,
               'tutt_1', 'tutt_2')
        # (above) creating button to go into next screen, Tutorial 2
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x() # Close window if 'X' button (top right) or
        # ESC key is pressed
```

b. tutorial_two()

```
def tutorial_two():
    '''Go to the Tutorial 2 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_2, (0, 0))
        # display image of Tutorial screen 2
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_2', 'tutt_1')
        # (above) creating button to go into the previous screen, Tutorial 1
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_2', 'tutt_3')
        # (above) creating button to go into next screen, Tutorial 3
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

c. tutorial_three()

```
def tutorial_three():
    '''Go to the Tutorial 3 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_3, (0, 0))
        # display image of Tutorial screen 3
```

```
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_3', 'tutt_2')
        # (above) creating button to go into the previous screen, Tutorial 2
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h,
               c.red_dim, c.red_lit, 'Next', button_font_size, c.black,
               'tutt_3', 'tutt_4')
        # (above) creating button to go into next screen, Tutorial 4
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

d. tutorial_four()

```
def tutorial_four():
    '''Go to the Tutorial 4 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_4, (0, 0))
        # display image of Tutorial screen 4
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_4', 'tutt_3')
        # (above) creating button to go into the previous screen, Tutorial 3
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_4', 'tutt_5')
        # (above) creating button to go into next screen, Tutorial 5
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

e. tutorial_five()

```
def tutorial_five():
    '''Go to the Tutorial 5 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_5, (0, 0))
        # display image of Tutorial screen 5
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_5', 'tutt_4')
        # (above) creating button to go into the previous screen, Tutorial 4
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_5', 'tutt_6')
        # (above) creating button to go into next screen, Tutorial 6
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

f. tutorial_six()

```
def tutorial_six():
    '''Go to the Tutorial 6 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_6, (0, 0))
        # display image of Tutorial screen 6
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_6', 'tutt_5')
        # (above) creating button to go into the previous screen, Tutorial 5
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_6', 'tutt_7')
        # (above) creating button to go into next screen, Tutorial 7
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

g. tutorial_seven()

```
def tutorial_seven():
    '''Go to the Tutorial 7 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_7, (0, 0))
        # display image of Tutorial screen 7
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_7', 'tutt_6')
        # (above) creating button to go into the previous screen, Tutorial 6
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_7', 'tutt_8')
        # (above) creating button to go into next screen, Tutorial 8
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

h. tutorial_eight()

```
def tutorial_eight():
    '''Go to the Tutorial 8 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_8, (0, 0))
        # display image of Tutorial screen 8
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_8', 'tutt_7')
        # (above) creating button to go into the previous screen, Tutorial 7
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_8', 'tutt_9')
        # (above) creating button to go into next screen, Tutorial 9
```

```
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

i. tutorial_nine()

```
def tutorial_nine():
    '''Go to the Tutorial 9 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_9, (0, 0))
        # display image of Tutorial screen 9
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_9', 'tutt_8')
        # (above) creating button to go into the previous screen, Tutorial 8
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_9', 'tutt_10')
        # (above) creating button to go into next screen, Tutorial 10
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

j. tutorial_ten()

```
def tutorial_ten():
    '''Go to the Tutorial 10 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_10, (0, 0))
        # display image of Tutorial screen 10
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_10', 'tutt_9')
        # (above) creating button to go into the previous screen, Tutorial 9
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_10', 'tutt_11')
        # (above) creating button to go into next screen, Tutorial 11
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

k. tutorial_eleven()

```
def tutorial_eleven():
    '''Go to the Tutorial 11 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_11, (0, 0))
        # display image of Tutorial screen 11
```

```
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_11', 'tutt_10')
        # (above) creating button to go into the previous screen, Tutorial 10
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_11', 'tutt_12')
        # (above) creating button to go into next screen, Tutorial 12
        ut.update_screen(clock) # Updating screen
        ut.quit_x() # Close window if 'X' button (top right) or ESC key is pressed
```

l. tutorial_twelve()

```
def tutorial_twelve():
    '''Go to the Tutorial 12 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_12, (0, 0))
        # display image of Tutorial screen 12
        button('top left', tutt_button_offset_x, tutt_raised_button_y_pos, button_w,
               button_h, c.red_dim, c.red_lit, 'Back', button_font_size, c.black,
               'tutt_12', 'tutt_11')
        # (above) creating button to go into the previous screen, Tutorial 11
        button('top right', display_width - tutt_button_offset_x,
               tutt_raised_button_y_pos, button_w, button_h, c.red_dim, c.red_lit,
               'Next', button_font_size, c.black, 'tutt_12', 'tutt_13')
        # (above) creating button to go into next screen, Tutorial 13
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

m. tutorial_thirteen()

```
def tutorial_thirteen():
    '''Go to the Tutorial 13 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_13, (0, 0))
        # display image of Tutorial screen 13
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_13', 'tutt_12')
        # (above) creating button to go into the previous screen, Tutorial 12
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_13', 'tutt_14')
        # (above) creating button to go into next screen, Tutorial 14
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

n. tutorial_fourteen()

```
def tutorial_fourteen():
    '''Go to the Tutorial 14 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_14, (0, 0))
        # display image of Tutorial screen 14
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_14', 'tutt_13')
        # (above) creating button to go into the previous screen, Tutorial 13
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_14', 'tutt_15')
        # (above) creating button to go into next screen, Tutorial 15
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

o. tutorial_fifteen()

```
def tutorial_fifteen():
    '''Go to the Tutorial 15 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_15, (0, 0))
        # display image of Tutorial screen 15
        button('top left', tutt_button_offset_x, tutt_raised_button_y_pos,
               button_w, button_h, c.red_dim, c.red_lit, 'Back', button_font_size,
               c.black, 'tutt_15', 'tutt_14')
        # (above) creating button to go into the previous screen, Tutorial 14
        button('top right', display_width - tutt_button_offset_x,
               tutt_raised_button_y_pos, button_w, button_h, c.red_dim, c.red_lit,
               'Next', button_font_size, c.black, 'tutt_15', 'tutt_16')
        # (above) creating button to go into next screen, Tutorial 16
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

p. tutorial_sixteen()

```
def tutorial_sixteen():
    '''Go to the Tutorial 16 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_16, (0, 0))
        # display image of Tutorial screen 16
        button('top left', tutt_button_offset_x, tutt_raised_button_y_pos,
               button_w, button_h, c.red_dim, c.red_lit, 'Back', button_font_size,
               c.black, 'tutt_16', 'tutt_15')
        # (above) creating button to go into the previous screen, Tutorial 15
```

```
        button('top right', display_width - tutt_button_offset_x,
               tutt_raised_button_y_pos, button_w, button_h, c.red_dim, c.red_lit,
               'Next', button_font_size, c.black, 'tutt_16', 'tutt_17')
        # (above) creating button to go into next screen, Tutorial 17
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

q. tutorial_seventeen()

```
def tutorial_seventeen():
    '''Go to the Tutorial 17 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_17, (0, 0))
        # display image of Tutorial screen 17
        button('top left', tutt_button_offset_x, tutt_raised_button_y_pos,
               button_w, button_h, c.red_dim, c.red_lit, 'Back', button_font_size,
               c.black, 'tutt_17', 'tutt_16')
        # (above) creating button to go into the previous screen, Tutorial 16
        button('top right', display_width - tutt_button_offset_x,
               tutt_raised_button_y_pos, button_w, button_h, c.red_dim, c.red_lit,
               'Next', button_font_size, c.black, 'tutt_17', 'tutt_18')
        # (above) creating button to go into next screen, Tutorial 18
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

r. tutorial_eighteen()

```
def tutorial_eighteen():
    '''Go to the Tutorial 18 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_18, (0, 0))
        # display image of Tutorial screen 18
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_18', 'tutt_17')
        # (above) creating button to go into the previous screen, Tutorial 17
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_18', 'tutt_19')
        # (above) creating button to go into next screen, Tutorial 19
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

s. tutorial_nineteen()

```
def tutorial_nineteen():
    '''Go to the Tutorial 19 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_19, (0, 0))
        # display image of Tutorial screen 19
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_19', 'tutt_18')
        # (above) creating button to go into the previous screen, Tutorial 18
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_19', 'tutt_20')
        # (above) creating button to go into next screen, Tutorial 20
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

t. tutorial_twenty()

```
def tutorial_twenty():
    '''Go to the Tutorial 20 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_20, (0, 0))
        # display image of Tutorial screen 20
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_20', 'tutt_19')
        # (above) creating button to go into the previous screen, Tutorial 19
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Next', button_font_size, c.black, 'tutt_20', 'tutt_21')
        # (above) creating button to go into next screen, Tutorial 21
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

u. tutorial_twentyone()

```
def tutorial_twentyone():
    '''Go to the Tutorial 21 screen with button to go to different screen'''
    while True:
        gameDisplay.blit(pic.tutorial_21, (0, 0))
        # display image of Tutorial screen 21
        button('bottom left', tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'tutt_21', 'tutt_20')
        # (above) creating button to go into the previous screen, Tutorial 20
```

```
        button('bottom right', display_width - tutt_button_offset_x,
               display_height - tutt_button_offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Main Menu', button_font_size, c.black, 'tutt_21', 'intro')
        # (above) creating button to go into Main menu
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

Credits Screen

- **Main**

a. credits_one()

```
def credits_one():
    while True:
        gameDisplay.blit(pic.credits_1, (0, 0))
        # display Credits 1 screen
        button('top left', offset_x, offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Main Menu', button_font_size, c.black, 'credits_one', 'intro')
        # (above) creating button to go into Main menu
        button('top right', display_width - offset_x, offset_y, button_w, button_h,
               c.green_dim, c.green_lit, 'Next', button_font_size, c.black, 'credits_one',
               'credits_two')
        # (above) creating button to go into Credits 2 screen
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

b. credits_two()

```
def credits_two():
    while True:
        gameDisplay.blit(pic.credits_2, (0, 0))
        # display Credits 2 screen
        button('top left', offset_x, offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'credits_two', 'credits_one')
        # (above) creating button to go into Credits 1 screen
        button('top right', display_width - offset_x, offset_y, button_w, button_h,
               c.green_dim, c.green_lit, 'Next', button_font_size, c.black, 'credits_two',
               'credits_three')
        # (above) creating button to go into Credits 3 screen
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

c. credits_three()

```
def credits_three():
    while True:
        gameDisplay.blit(pic.credits_3, (0, 0))
        # display Credits 3 screen
        button('top left', offset_x, offset_y, button_w, button_h, c.red_dim,
               c.red_lit, 'Back', button_font_size, c.black, 'credits_three', 'credits_two')
        # (above) creating button to go into Credits 2 screen
        button('top right', display_width - offset_x, offset_y, button_w, button_h,
               c.red_dim, c.red_lit, 'Main Menu', button_font_size, c.black, 'credits_three',
               'intro')
        # (above) creating button to go into Main menu
        ut.update_screen(clock)
        # Updating screen
        ut.quit_x()
        # Close window if 'X' button (top right) or ESC key is pressed
```

Utilities

- **Button**
  a. button(placement, x, y, w, h, default_color, light_up_color, msg, text_size, text_color, previous, action=None, font_type='freesansbold.ttf')

```
# Function to create button with text
def button(placement, x, y, w, h, default_color, light_up_color, msg, text_size,
           text_color, previous, action=None, font_type='freesansbold.ttf'):
    '''create button'''
    global cap
    # var. to get the video capture

    if placement == 'center':
        # if x and y (param.) coordinates is to be at the center of the button,
        x = x - w//2
        y = y - h//2

    elif placement == 'top right':
        # if x and y (param.) coordinates is to be at the top right of the button,
        x = x - w

    elif placement == 'bottom left':
        # if x and y (param.) coordinates is to be at the bottom left of the button,
        y = y - h

    elif placement == 'bottom right':
        # if x and y (param.) coordinates is to be at the bottom right of the button,
        x = x - w
        y = y - h

    mouse = pygame.mouse.get_pos()
    # get mouse position (coordinate) and store it into mouse var.
    if x < mouse[0] < x+w and y < mouse[1] < y+h:
        # if mouse is inside the button,
```

```
            pygame.draw.rect(gameDisplay, light_up_color, (x, y, w, h))
            # change the color of the button as if it lit up
            for event in pygame.event.get():
                # check for any event
                if event.type == pygame.MOUSEBUTTONDOWN:
                    # if clicked,
                    if previous == 'game':
                        # if the screen before next screen is the game (playing) screen
                        cap.release()
                        # release the camera from capturing frames
                    if action:
                        # if there is action,
                        if action == 'paused':
                            # if next screen is pause screen,
                            sound.pause()
                            # sound the pause button sound
                            pause_menu()
                            # go into the Pause menu

                            # in the Pause menu,
                            pygame.mixer.music.set_volume(0.3)
                            # turn down the volume of the music previously played
                        else:
                            pygame.mixer.music.set_volume(1.0)
                            # other than Pause menu, turn up the volume to normal volume
                            if action == 'game':
                                # if next screen is Game screen,
                                if scanned == 1:
                                    # if hand is scanned,
                                    sound.play()
                                    # sound the game button sound
```

```
                                    game_screen()
                                    # go into Game screen
                                else:
                                    pause_scan_first()
                                    # if hand is not scanned, show the scan first message
                            elif action == 'intro':
                                # if next screen is Main menu,
                                sound.select()
                                # sound the default button sound
                                intro()
                                # go into Main menu
                            elif action == 'quit':
                                # if quitting the game,
                                ut.quit_game()
                                # close window

                            elif action == 'scan_screen':
                                # if next screen is Scan screen,
                                sound.play()
                                # sound the scan/play button sound
                                music.stop()
                                # stop the previous music
```

```
                music.game()
                # play the music for scan/play screen
                hand_scan_screen()
                # go into the Scan screen
            elif action == 'hand_scan':
                # if pressed 'Scan' button to scan the hand
                # positioned in screen,
                sound.play()
                # sound the scan/play button sound
                return True
                # return True to indicate that the hand has been scanned
            elif action == 'tutt_1':
                # if next screen is Tutorial 1 screen,
                if previous  == 'intro' or previous == 'paused':
                    # if the screen previously is Main or Pause menu,
                    music.stop()
                    # stop the music played previously
                    music.instruc()
                    # play the music for Instruction screen
                    sound.select()
                    # sound the default button sound
                    tutorial_one()
                    # go into the Tutorial 1 screen
                else:
                    sound.select()
                    # sound the default button sound
                    tutorial_one()
                    # go into the Tutorial 1 screen
            elif action == 'tutt_2':
                # if next screen is Tutorial 2 screen,
```

```
                sound.select()
                # sound the default button sound
                tutorial_two()
                # go into the Tutorial 2 screen
            elif action == 'tutt_3':
                # if next screen is Tutorial 3 screen,
                sound.select()
                # sound the default button sound
                tutorial_three()
                # go into the Tutorial 3 screen
            elif action == 'tutt_4':
                # if next screen is Tutorial 4 screen,
                sound.select()
                # sound the default button sound
                tutorial_four()
                # go into the Tutorial 4 screen
            elif action == 'tutt_5':
                # if next screen is Tutorial 5 screen,
                sound.select()
                # sound the default button sound
                tutorial_five()
                # go into the Tutorial 5 screen
            elif action == 'tutt_6':
                # if next screen is Tutorial 6 screen,
```

```
                sound.select()
                # sound the default button sound
                tutorial_six()
                # go into the Tutorial 6 screen
            elif action == 'tutt_7':
                # if next screen is Tutorial 7 screen,
                sound.select()
                # sound the default button sound
                tutorial_seven()
                # go into the Tutorial 7 screen
            elif action == 'tutt_8':
                # if next screen is Tutorial 8 screen,
                sound.select()
                # sound the default button sound
                tutorial_eight()
                # go into the Tutorial 8 screen
            elif action == 'tutt_9':
                # if next screen is Tutorial 9 screen,
                sound.select()
                # sound the default button sound
                tutorial_nine()
                # go into the Tutorial 9 screen
            elif action == 'tutt_10':
                # if next screen is Tutorial 10 screen,
                sound.select()
                # sound the default button sound
                tutorial_ten()
                # go into the Tutorial 10 screen
            elif action == 'tutt_11':
                # if next screen is Tutorial 11 screen,
```

```
                sound.select()
                # sound the default button sound
                tutorial_eleven()
                # go into the Tutorial 11 screen
            elif action == 'tutt_12':
                # if next screen is Tutorial 12 screen,
                sound.select()
                # sound the default button sound
                tutorial_twelve()
                # go into the Tutorial 12 screen
            elif action == 'tutt_13':
                # if next screen is Tutorial 13 screen,
                sound.select()
                # sound the default button sound
                tutorial_thirteen()
                # go into the Tutorial 13 screen
            elif action == 'tutt_14':
                # if next screen is Tutorial 14 screen,
                sound.select()
                # sound the default button sound
                tutorial_fourteen()
                # go into the Tutorial 14 screen
```

```
                elif action == 'tutt_15':
                    # if next screen is Tutorial 15 screen,
                    sound.select()
                    # sound the default button sound
                    tutorial_fifteen()
                    # go into the Tutorial 15 screen
                elif action == 'tutt_16':
                    # if next screen is Tutorial 16 screen,
                    sound.select()
                    # sound the default button sound
                    tutorial_sixteen()
                    # go into the Tutorial 16 screen
                elif action == 'tutt_17':
                    # if next screen is Tutorial 17 screen,
                    sound.select()
                    # sound the default button sound
                    tutorial_seventeen()
                    # go into the Tutorial 17 screen
                elif action == 'tutt_18':
                    # if next screen is Tutorial 18 screen,
                    sound.select()
                    # sound the default button sound
                    tutorial_eighteen()
                    # go into the Tutorial 18 screen
                elif action == 'tutt_19':
                    # if next screen is Tutorial 19 screen,
                    sound.select()
                    # sound the default button sound
                    tutorial_nineteen()
```

```
                    # go into the Tutorial 19 screen
                elif action == 'tutt_20':
                    # if next screen is Tutorial 20 screen,
                    sound.select()
                    # sound the default button sound
                    tutorial_twenty()
                    # go into the Tutorial 20 screen
                elif action == 'tutt_21':
                    # if next screen is Tutorial 21 screen,
                    sound.select()
                    # sound the default button sound
                    tutorial_twentyone()
                    # go into the Tutorial 21 screen
                elif action == 'credits_one':
                    # if next screen is Credit 1 screen,
                    if previous_ == 'intro':
                        music.stop()
                        # stop the music playing previously
                        music.credits()
                        # play the music for credits
                        sound.select()
                        # sound the default button sound
                        credits_one()
                        # go into the Credits 1 screen
```

```
                    else:
                        sound.select()
                        # sound the default button sound
                        credits_one()
                        # go into the Credits 1 screen
                elif action == 'credits_two':
                    # if next screen is Credit 2 screen,
                    sound.select()
                    # sound the default button sound
                    credits_two()
                    # go into the Credits 2 screen
                elif action == 'credits_three':
                    # if next screen is Credit 3 screen,
                    sound.select()
                    # sound the default button sound
                    credits_three()
                    # go into the Credits 3 screen

    else:
        # if mouse is not in the button,
        pygame.draw.rect(gameDisplay, default_color, (x, y, w, h))
        # do not change the color of button

    if not msg == '':
        # if there is message to write in the button,
        button_text(msg, text_size, text_color, font_type, x, y, w, h)
```

b. button_text(msg, text_size, color, font_type, x, y, w, h)

```
# Function to create text for button
def button_text(msg, text_size, color, font_type, x, y, w, h):
    '''create font object for button'''
    font = pygame.font.Font(font_type, text_size)
    # create a new font object
    textSurf, textRect = text_object(msg, font, color)
    # get text surface object and rectangular area of the surface
    textRect.center = (x + w // 2, y + h // 2)
    # get the center of the text to be at the center of the button
    gameDisplay.blit(textSurf, textRect)
    # display text
```

- **Text**
  a. text(msg, text_size, color, x, y, positioning = '', back_color=None, font_type='freesansbold.ttf')

```
def text(msg, text_size, color, x, y, positioning = '', back_color=None,
         font_type='freesansbold.ttf'):
    '''create text and display it'''
    font = pygame.font.Font(font_type, text_size)
    # create a new font object

    textSurf, textRect = text_object(msg, font, color, back_color)
    # get text surface object and rectangular area of the surface
```

```
    if positioning == 'center of box':
        # if the text is to be in the center of the box,
        textRect.center = (x // 2, y // 2)
    elif positioning == 'center point':
        # if text is to be positioned using the center point of the text,
        textRect.center = (x, y)
    else:
        # if text is to be positioned using the top left coordinate of the text,
        textRect = (x, y)

    gameDisplay.blit(textSurf, textRect)
    # draw image in this case text
```

b. text_object(msg, font, color, back_color=None)

```
# Functions to create and display text
def text_object(msg, font, color, back_color=None):
    '''create text surface object, where text is drawn on it
    return text surface and rectangle area of the text'''

    textSurface = font.render(msg, True, color, back_color)
    # create surface with text drawn on it
    return textSurface, textSurface.get_rect()
    # return text surface object and rectangular area of the text surface
```

c. text_size(msg, font_size, font_type='freesansbold.ttf')

```
# Function to get the width and the height of the text
def text_size(msg, font_size, font_type='freesansbold.ttf'):
    '''Returns the width and the height of the text'''
    txt = pygame.font.Font(font_type, font_size)
    txt_w, txt_h = txt.size(msg)[0], txt.size(msg)[1]
    return txt_w, txt_h
```

Music Program (music.py)

a. intro()

```
def intro():
    '''Music for Main Menu'''
    music = random.choice(Intro)
    # randomly choose a music from Music/Intro folder
    pygame.mixer.music.load('Music/Intro/'+music)
    # load music
    pygame.mixer.music.play(-1)
    # play music infinitely
```

b. instruc()

```
def instruc():
    '''Music for Instructions'''
    music = random.choice(Instruction)
    # randomly choose a music from Music/Instruction folder
    pygame.mixer.music.load('Music/Instruction/'+music)
    # load music
    pygame.mixer.music.play(-1)
    # play music infinitely
```

c. game()

```
def game():
    '''Music for Game Screen'''
    music = random.choice(Game)
    # randomly choose a music from Music/Game folder
    pygame.mixer.music.load('Music/Game/'+music)
    # load music
    pygame.mixer.music.play(-1)
    # play music infinitely
```

d. win()

```
def win():
    '''Music for Result Screen (Win)'''
    music = random.choice(Win)
    # randomly choose a music from Music/Win folder
    pygame.mixer.music.load('Music/Win/'+music)
    # load music
    pygame.mixer.music.play(-1)
    # play music infinitely
```

e. lose()

```
def lose():
    '''Music for Result Screen (Lose)'''
    music = random.choice(Lose)
    # randomly choose a music from Music/Lose folder
    pygame.mixer.music.load('Music/Lose/'+music)
    # load music
    pygame.mixer.music.play(-1)
    # play music infinitely
```

f. draw()

```
def draw():
    '''Music for Result Screen (Draw)'''
    music = random.choice(Draw)
    # randomly choose a music from Music/Draw folder
    pygame.mixer.music.load('Music/Draw/'+music)
    # load music
    pygame.mixer.music.play(-1)
    # play music infinitely
```

g. credits()

```
def credits():
    '''Music for Credits screen'''
    music = random.choice(Credits)
    # randomly choose a music from Music/Credits folder
    pygame.mixer.music.load('Music/Credits/'+music)
    # load music
    pygame.mixer.music.play(-1)
    # play music infinitely
```

h. stop()

```
def stop():
    '''To stop music that is playing'''
    pygame.mixer.music.stop()
    # stop playing music
```

Sound Program (sound.py)

a. select()

```
def select():
    ''' Play default button press sound'''
    pressing_sound = pygame.mixer.Sound('Sound/NFF-select-04.wav')
    # load sound
    pygame.mixer.Channel(0).play(pygame.mixer.Sound(pressing_sound))
    # play sound on Channel 0
```

b. pause()

```
def pause():
    ''' Play 'Pause' button press sound'''
    pressing_sound = pygame.mixer.Sound('Sound/NFF-menu-04-a.wav')
    # load sound
    pygame.mixer.Channel(0).play(pygame.mixer.Sound(pressing_sound))
    # play sound on Channel 0
```

c. play()

```
def play():
    ''' Play 'Play', 'Play again', 'Scan' or 'Scan again' button press sound'''
    pressing_sound = pygame.mixer.Sound('Sound/NFF-menu-a.wav')
    # load sound
    pygame.mixer.Channel(0).play(pygame.mixer.Sound(pressing_sound))
    # play sound on Channel 0
```

d. scan_first()

```
def scan_first():
    ''' Play this sound when the Scan first message is shown in the Pause menu'''
    pressing_sound = pygame.mixer.Sound('Sound/NFF-funny-cancel.wav')
    # load sound
    pygame.mixer.Channel(0).play(pygame.mixer.Sound(pressing_sound))
    # play sound on Channel 0
```

e. win()

```
def win():
    ''' Play this sound when user wins in the game screen'''
    pressing_sound = pygame.mixer.Sound('Sound/NFF-gong.wav')
    # load sound
    pygame.mixer.Channel(0).play(pygame.mixer.Sound(pressing_sound))
    # play sound on Channel 0
```

f. lose()

```
def lose():
    ''' Play this sound when user lose in the game screen'''
    pressing_sound = pygame.mixer.Sound('Sound/NFF-lose.wav')
    # load sound
    pygame.mixer.Channel(0).play(pygame.mixer.Sound(pressing_sound))
    # play sound on Channel 0
```

g. draw()

```
def draw():
    ''' Play this sound when draw in the game screen'''
    pressing_sound = pygame.mixer.Sound('Sound/NFF-prompt-soft.wav')
    # load sound
    pygame.mixer.Channel(0).play(pygame.mixer.Sound(pressing_sound))
    # play sound on Channel 0
```

h. timing()

```
def timing():
    ''' Play this sound when the circles in the game screen lit into orange color'''
    pressing_sound = pygame.mixer.Sound('Sound/NFF-ethno-confirmation.wav')
    # load sound
    pygame.mixer.Channel(0).play(pygame.mixer.Sound(pressing_sound))
    # play sound on Channel 0
```

Utils

a. mouse_pos()

```
def mouse_pos():
    ''' Returns mouse coordinate'''
    return pygame.mouse.get_pos()
```

b. update_screen(clock)

```
def update_screen(clock):
    ''' Updates screen with 60 FPS'''
    pygame.display.update()
    clock.tick(60)
```

c. quit_x()

```
def quit_x():
    ''' Quits game when 'x' button or ESC key is pressed'''
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            # if 'x' button is pressed,
            # (below) quits game
            pygame.quit()
            quit()
            cap.release()
            # release webcam from capture
        elif event.type == pygame.KEYDOWN:
            # if any key is pressed,
            if event.key == pygame.K_ESCAPE:
                # if ESC key is pressed,
                # (below) quits game
                pygame.quit()
                quit()
                cap.release()
                # release webcam from capture
```

d. quit_game()

```
def quit_game():
    ''' Quits game'''
    pygame.quit()
    quit()
    cap.release()
    # release webcam from capture
```

Conclusion

Accomplishment

At the end of this Final Year Project, I was able to create a software that can:

- Recognise the gesture of the user (Scissor, Paper, Stone) through a webcam with more than 95% accuracy with restrictions. The restrictions include: the user is required to scan their hand properly; the user must show their gesture's front view only; do not include anything other than the user's hand in the green box drawn on the video stream of the Game screen.
- Display user's gesture that is predicted by the trained CNN model.
- Respond to the user's gesture with a randomly chosen computer gesture and display both sides' gesture within 10 milliseconds.
- Engage with the user through GUI which is easy to navigate through the different screens of the game.

Further Enhancement

The software has problems:

- Segmenting the skin when strong lighting is directly facing the webcam.
- Predicting the user's gesture accurately when the user does not show the front of the gesture. Scissor Paper Stone is usually not done with the gesture front view facing the opponent as it would be troublesome to adjust for the user.

Suggested improvements:

- Change the position of the webcam. Instead of directly facing the user, it could face sideways to the user to detect user's gesture from the side. That way, the user does not need to show the gesture front view. Instead the user can show their gesture naturally like how the game is usually played with people.
- Have two webcams to detect the gesture. Using two webcams would increase the data of the user's gesture, increasing the accuracy of the prediction.

Appendices

Appendices A

Detection speed using Nvidia GeForce GTX TITAN X card

COCO-trained models

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_mobilenet_v2_quantized_coco	29	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks

Reference

May 7, 2015. Finger Tracking with OpenCV and Python.

http://www.benmeline.com/finger-tracking-with-opencv-and-python/

(Last Accessed: 3 June 2019)

Chin Huan Tan. Real-time Finger Detection.

https://becominghuman.ai/real-time-finger-detection-1e18fea0d1d4

(Last Accessed: 3 June 2019)

Francois Chollet. Building powerful image classification models using very little data.

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

(Last Accessed: 3 June 2019)

Introduction to PyGame.

https://pythonprogramming.net/pygame-python-3-part-1-intro/

(Last Accessed: 1 June 2019)

Displaying images with PyGame.

https://pythonprogramming.net/displaying-images-pygame/

(Last Accessed: 1 June 2019)

Displaying text to PyGame screen.

https://pythonprogramming.net/displaying-text-pygame-screen/

(Last Accessed: 1 June 2019)

Drawing objects with PyGame.

https://pythonprogramming.net/drawing-objects-pygame-tutorial/

(Last Accessed: 1 June 2019)

Drawing Objects and Shapes in PyGame.

https://pythonprogramming.net/pygame-drawing-shapes-objects/

(Last Accessed: 1 June 2019)

PyGame Buttons, part 1, drawing the rectangle.

https://pythonprogramming.net/pygame-buttons-part-1-button-rectangle/

(Last Accessed: 1 June 2019)

PyGame Buttons, part 2, making the buttons interactive.

https://pythonprogramming.net/making-interactive-pygame-buttons/

(Last Accessed: 1 June 2019)

PyGame Buttons, part 3, adding text to the button

https://pythonprogramming.net/placing-text-pygame-buttons/

(Last Accessed: 1 June 2019)

PyGame Buttons, part 4, creating a general PyGame button function.

https://pythonprogramming.net/pygame-button-function/

(Last Accessed: 1 June 2019)

PyGame Buttons, part 5, running functions on a button click.

https://pythonprogramming.net/pygame-button-function-events/

(Last Accessed: 1 June 2019)

Sounds and Music with PyGame.

https://pythonprogramming.net/adding-sounds-music-pygame/

(Last Accessed: 1 June 2019)