JANUARY 24, 2020

Flinders
UNIVERSITY

NANYANG
POLYTECHNIC

FINAL REPORT

INTERNATIONAL PRACTICUM | NANYANG POLYTECHNIC

STUDENT: KARAN VARMA
BACHELOR'S IN ROBOTIC ENGINEERING | FLINDERS UNIVERSITY
ACADEMIC SUPERVISOR: DR. NASSER ASGARI | FLINDERS UNIVERSITY
WORKPLACE SUPERVISOR: DR FU TUAN HOE EDWIN | NANYANG POLYTECHNIC
PLACEMENT START DATE: 2/09/2019
PLACEMENT END DATE: 24/01/2020
SUBMISSION DATE: 31/01/2020

Declaration

I certify that this work does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Karan Varma

Karan Varma
January 2020

Abstract

This report covers the project undertaken at Nanyang Polytechnic during the 20-week placement. Phase one aims to improve upon a 'Rock Paper Scissors' game previously made. The game uses a webcam to see the users hand gesture and picks its own gesture, the result is then displayed. The game has a number of bugs and issues that need to be improved upon. The second phase aims to work on programming an autonomous robot which can navigate around the building. This robot will carry the game module and stop if users want play the game. By achieving the above, Nanyang Polytechnic will be able to provide students with a fun and engaging experience which should lead to a higher student interest within robotic engineering.

During the project timeline various issues were face, the biggest of them was the motor driver for the robot being faulty. This happened in week 12 and resulted in the project scope being changed since there would not be enough time to complete the second phase of the project. Hence, the project was changed to only the first phase from the initial project. The extra time allowed for the game to be improved upon significantly. The improvements made are all discussed within the report. At the end of the project, the game functioned very well, all the bugs/issues were removed and the code was approximately 500 lines shorter. The complete source code is provided under the appendix B.

Overall the project was a success and Nanyang Polytechnic was pleased with the results achieved despite the significant changes to the project due to the faulty parts which was an uncontrollable issue. Undertaking the project provided me with lots of experience and new skills were developed. Skills such as Python programming, time management, working individually with little supervision as well as communication with stakeholders were developed. The project is deemed successful for all involved.

Table of Contents

List of Tables

List of Figures

1: Introduction

Nanyang Polytechnic is located in in the precinct of Yio Chu Kang, Singapore and was established in 1992. The polytechnic has a range of schools such as business management, life sciences, engineering, information technology and design (Polytechnic).

The focus of this report is the project undertaken which required for a 'Rock Paper Scissors' game to improved. The game was previously developed by a student named Oei Zheng Yong but requires improvement in many areas. Initially, the project also aimed to program an autonomous robot, however, due to faulty parts this part of the project was left out.

This report outlines the project, its scope, purpose, hardware and software specifications, the assumptions and constraints, design, execution as well as results and an evaluation. During the timeline of this project some unforeseen changes were required to be made and these are also discussed in detail.

The project objectives are as follows:

- Improve Game Graphical User Interface (GUI).
- Resolve the issues the game has.
- Improve the gesture recognition accuracy.
- Decrease the game's memory consumption by removing redundant code and using smarter programming techniques.
- Make the game more entertaining for users.

2. Project Vision

In recent years, the demand for robotic automation within industries is rapidly increasing. This is vastly due to the many advantages of robots including the following:

- High productivity with long operation hours.
- Resolve shortage of skill labour issue.
- Flexible operations.
- Consistent quality in production.
- Able to complete repetitive and tedious tasks with ease.
- Able to work in hazardous environment.

As the demand for automation within the industry increases, the need for engineers to develop and troubleshoot robots is also increasing. Hence, to cope with the demand, we need more students to develop interest in robotics and specialise in the field.

This project aims improve upon the 'Rock Paper Scissor' game developed as well as programming autonomous robot which can navigate by Itself and carry the game module so it can be played by individuals. By doing so, this project will help students develop interest within robotic engineering through providing an engaging and interactive game experience with an autonomous robot. The game programming will be completed by myself and the autonomous robot component of the project will be completed by Luen Zhe Yuan and myself.

3. Project Background

A 'Rock Paper Scissor' game has been previously developed. This game uses a video stream to recognise the user's gesture and responds with a gesture picked by the computer. Figure 1 illustrates the basic game setup.

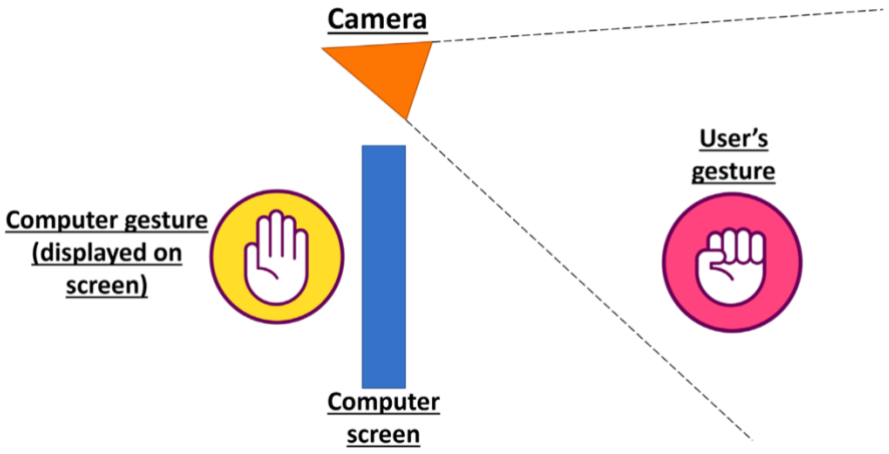


Figure 1: Illustration of the game setup.

The game has the following game modes:

- Easy
- Intermediate
- Hard
- Expert

The easy mode works by randomly picking the computers choice of gesture. The intermediate mode is a little bit smarter and tries to be less obvious. The Hard mode uses machine learning to have a better chance at beating the user. Lastly, the expert mode predicts the hand gesture midway by checking the video stream and selects the best gesture to win. The computer rarely loses in expert mode. The figures 2 illustrates how the user's hand colour is scanned and figure 3 illustrates the game screen.

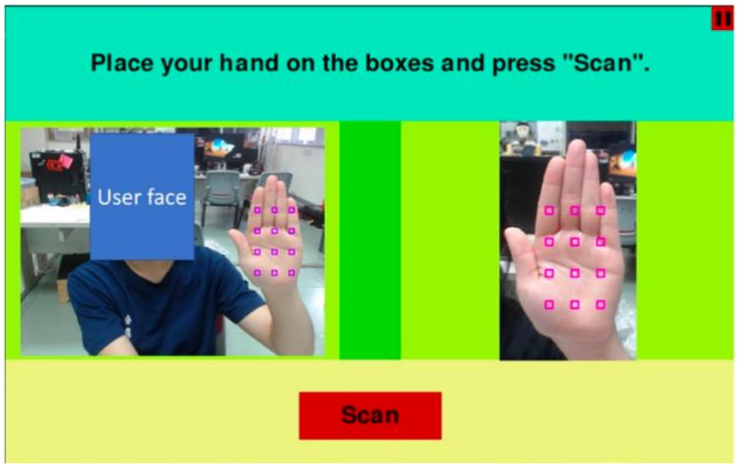


Figure 2: Original game GUI - Hand Scan Screen.

Figure 3: Original game GUI - Match Screen.

3.1 Software

3.1.1 Operating System

The operating system used is Ubuntu 18.04. Ubuntu is a free and open-source Linux distribution based on Debian. It is used for the 'Rock Paper Scissor' game and programming the autonomous robot.

3.1.2 Programming Language

The 'Rock Paper Scissor' game was developed using python 3 and a module named Pygame. Python 3 is a programming language that allows greater productivity and easier integration of systems. Pygame is a Python programming library which is used for multimedia applications.

3.1.3 Deep Learning

For the deep learning, Keras and TensorFlow have been previously used. 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 (CNN) model to recognise the user gesture.

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 to recognise the user gesture using TensorFlow backend.

3.1.4 Computer Vision

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.

3.2 Hardware

3.2.1 NVIDIA Jetson Nano

The Jetson Nano developer kit is a computer with a Quad-Core ARM 57 CPU, a 128-Core Maxwell GPU and 4GB LPDDR4 memory. The kit also has various input and output port such as HDMI, USB 2.0, USB 3.0 and microSD card slot. The developer kit is depicted below.

Figure 4: Illustration of the Jetson Nano Developer Kit.

3.2.2 Logitech HD Pro Webcam C910 (1080p)

This webcam is used to capture the video stream for the GUI, processing the image and getting the frame with the user's gesture.

3.2.3 Robot Hardware

The motor and motor controller chosen for the robot are manufactured by ZLTECH and are designed to work together. The motor features an encoder and the motor controller features Pulse Width Modulation (PWM). The figure below illustrates the motor and motor controller.

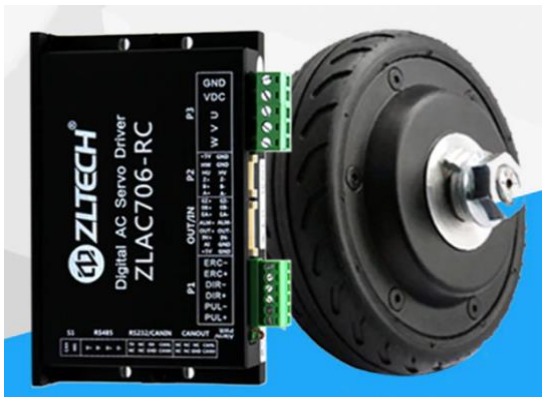


Figure 5: Illustration of the motor and motor controller (Express).

The robot frame provided is illustrated below:

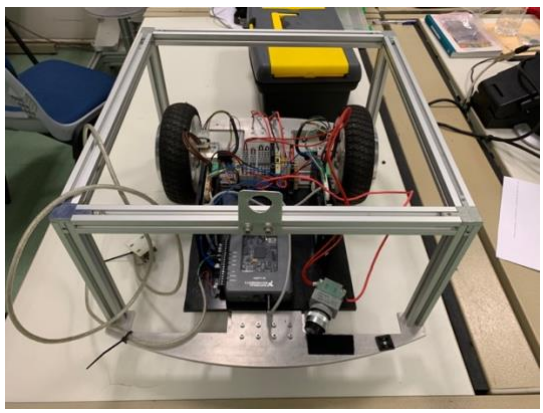


Figure 6: Illustration of the robot's frame.

3.3 Improvement Specifications

After a few meetings with the supervisor, the following game improvements were specified.

- The game requires an improved GUI.
- There are various bugs such as randomly crashing, and the mouse click not registering.
- Too much memory is being used. Hence, the game does not start until the system is rebooted. Often multiple reboots are required to start the game.
- The gesture recognition accuracy requires improvement.
- The loading time when transitioning from one game screen to another has a significant delay and requires improvements.
- The game window cannot be resized.
- The game needs to be more fun and engaging.

4. Assumptions and Constraints

The initial assumptions made for this project are presented below.

4.1 Resource Assumptions

It is assumed that the required resources will be available and will allow for the project to be continued with a maximum delay of one day. Allowing for such a delay throughout the time frame of the project will possibly allow for more considerable delays in different stages if previous stages did not have any delays. Accounting for possible delays helps lower the risk of compromised timeframes.

4.2 Hardware Assumptions

- It is assumed that the required hardware components will be available to allow for the project to continue. If not available, it is assumed a reasonable replacement will be found.
- It is also assumed that all hardware components will function correctly, and no significant hardware issues will be faced.

4.3 Technology Based Assumptions

- It is assumed that the required software will be available to allow for the project to continue. If not available, it is assumed a reasonable replacement will be found.
- It is assumed that the project team has the knowledge and capability to use the software to achieve the desired result.

4.4 Time Based Assumptions

The two phases of the project are expected to be finished by the end of placement (24th January).

4.5 Financial Assumptions

It is also assumed that necessary funding will be received to enable the student to successfully reach the desired outcome.

4.6 If Assumptions Are Incorrect

If any of the above assumptions are incorrect, they will need to be documented, corrected and communicated. If assumptions change during the project, the reasoning for change, the impact on the project as well as what adjustments need to be made will assessed.

4.7 Cost Constraints

- The budget for this project is limited.
- The product requires to be fault free as it will be used by a lot of people and needs to provide a good experience.
- Limited experience in Python game development and deep learning for gesture recognition.
- The project timeframe is limited.

4.8 Design Constraints

The game needs to be designed to run on the Jetson Nano Developer Kit, this means there is limited processing power available to be used by the game.

4.9 Time Constraints

The entire project needs to be completed within the 20-week placement.

5. Project Timeline

5.1 Unforeseen Changes

Initially the project had two phases. The first phase required making improvements to the 'Rock Paper Scissors' game and the second phase required programming a robot to navigate around the building. The robot would carry the game module and user's would be able to stop and play the game if they wished to.

However, during the 12th week of placement, the motor controller for the robot stopped functioning. Hence, they had to be sent back to the manufacturer in China. This meant it would take 3 - 4 weeks for the motor controller to reach the manufacturer, be repaired and be sent back to Nanyang Polytechnic. Due to this uncontrollable delay, the supervisor removed the second phase from the project as there wouldn't be enough time to complete it. Due to these significant changes in project scope much more time could be spent on making improvements to the game.

The table below presents the updated WBS created in week 12 of placement along with the estimated dates for each task/milestone to be started and finished.

Table 1: Updated WBS.

WBS	Name/Title	Type
1.0	Placement	Project

1.1	Research Discovery	Task
1.1.1	Define project scope.	Task
1.1.2	Interview stakeholders.	Task
1.1.3	Finalize the project scope.	Milestone
1.1.4	Define the requirements.	Task
1.2	Programming the 'Rock Paper Scissor' game	Alone
1.2.1	Resolve the bugs and issues with the game.	Milestone
1.2.2	Improve the game GUI.	Task
1.2.3	Add Sounds and animations.	Task
1.2.4	Improve the gesture recognition accuracy.	Task
1.2.5	Lower the memory usage.	Task
1.2.6	Speed up the game loading times.	Task
1.2.7	Add window resizing.	Task
1.3	Testing	Teamwork
1.3.1	Test the two systems working together.	Task
1.3.2	Get user feedback.	Task
1.3.3	Make suggested improvements.	Task
1.4	Rollout	Teamwork
1.4.1	Develop User Guide.	Task

6. Implementation and Execution

6.1 Development Strategies

6.1.1 Research

Research is a key component of this project as it is important to learn Pygame programming techniques in order to understand the previously written code and make further improvements to it. Research will be conducted online through programming tutorials and forum discussions. Due to lack of Pygame programming experience, research will be vital for the project to be successful over the duration of the placement.

6.1.2 Code Exploration

Understanding the previously written code very well is a very important step in order to make improvements to the game. Analysing the structure of the code will also help find the room for improvement in code structure and redundancy.

6.1.3 Testing and Modification

Testing the game as each improvement is being made is vital in order to ensure the game functions correctly. The code is quite large and regular testing is important to avoid debugging any new issues that may arise during the programming process.

6.2 Method Taken

This section will cover each of the tasks/milestones within the WBS and explain how they were achieved.

6.2.1 Improving the Game GUI

The original GUI for the game looked a bit crowded in some of the screens and the buttons never functioned on the first click, they always required multiple clicks to respond. The screen size was fixed and could not be altered, this was another GUI improvement required.

Firstly, new background images were sourced from the web. Two background images were chosen. One image for the main screen and results screen, the other for the game screens. The one for the game screen was specifically chosen to be a plain one to make it simpler for the user. Pygame.org was referred to, to learn how to display images within Pygame and also used to learn how to write code for buttons properly. After lots of debugging the issue with buttons was resolved, by using an event pump to ensure the button clicks are captured. The

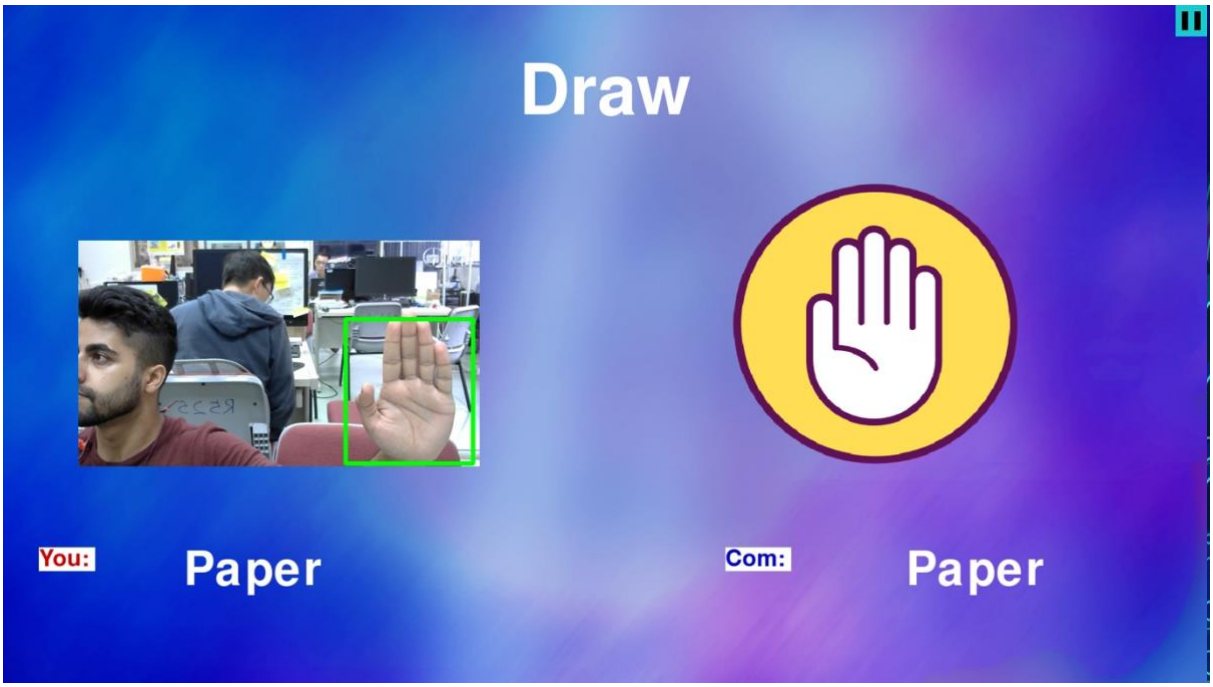


Figure 7: Improved Game GUI - Game Screen.

improved GUI is presented in the figures below, the complete GUI images can be found under appendix A.

6.2.2 Resizable Display

Making the window resizable took a rather long time as forums had to be searched to find the best way to achieve resizing. Upon finding the right method to make resizing possible, the next step was to alter the way font/image sizes are defined. All fonts/image sizes had to be redeclared relative to the window size and required to be updated every time the screen size changes. The function created to rescale the window is presented in figure 9 below.

```
def resizeDisplay():
    global display_width, display_height, button_w, button_h, text_size, text_size_1
    pygame.event.pump()
    event=pygame.event.wait()
    if event.type==QUIT: quit_all()
    elif event.type==VIDEORESIZE:
        display_width = event.w
        display_height = event.h
        button_w = int(200*(display_width/1200))
        button_h = int(50*(display_height/700))
        text_size = int(30*((display_width+display_height)/1900))
        text_size_1 = int(75*((display_width+display_height)/1900))
        gameDisplay = pygame.display.set_mode((display_width, display_height), HWSURFACE|DOUBLEBUF|RESIZABLE)
        pygame.display.flip()
        return display_width, display_height
```

Figure 9: The 'resizeDisplay' subroutine.

Figure 10 presents the resizable display feature functioning.

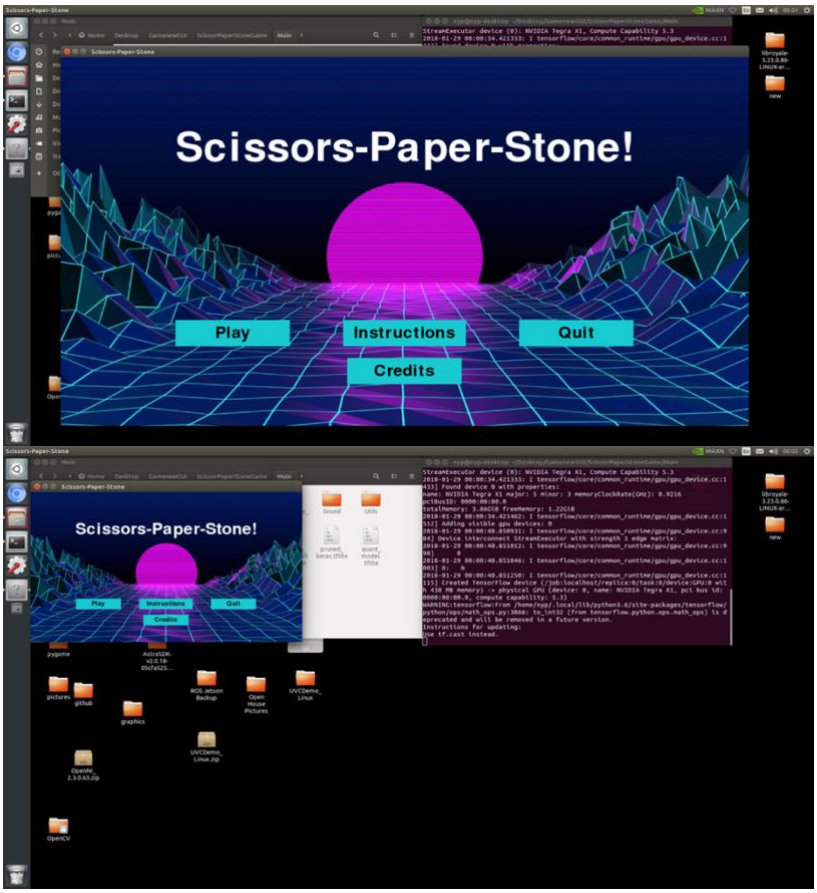


Figure 10:The resizable window size.

6.2.3 Making the Game Engaging

Upon discussing with the supervisor, the best way to make the game more engaging was to add more sounds to it. Background soundtracks along with sound effects upon winning/losing each round would make the game more entertaining. Pygame.org was referred to, for learning how sounds are played within Pygame and multiple background music as well as sound effects were sourced from online websites. Extra background soundtracks were downloaded so that they can be shuffled and different soundtracks can be played everytime the game starts. The soundtracks were then declared within a 'sound.py' file and used within the main source code.

6.3.4 Lowering memory usage and removing bugs

To achieve this objective, the 1873-line long code had to be analysed in detail to find redundant variables and lines of code. Pygame.org and forums were also consulted to find smarter ways of achieving the same results. This process took a large proportion of the project timeline due to lack of knowledge within Pygame programming. However, over a number of weeks, great progress was made. The game code was restructured in many ways and a lot of redundant code was removed. The 1873-line long code was shortened into only 1280 lines and the 1280-line long code had more feature and no bugs.

As an example, a large proportion of the redundant code was found within the subroutines for screen displays. Each of these displays functions had minor differences and through smarter structuring of code, these were comined and redundant code was eliminated. By removing redundant code, the original memory consumption could be lowered from ~2.7 GB down to ~2.2 GB.

Despite the lower memory consumption, the game still could not be restarted without rebooting the Jetson Nano. This issue, took some time to resolve. After a lot of research through forums, it was found that PyGame does not shut down all the processes if only 'pygame.quit()' is used to exit the game. Hence, a subroutine was created to ensure everything running within the game is quit to clear all the memory so the game can be restarted without rebooting the Jetson Nano. Figure 11 presents the 'quit_all' subroutine.

```
#quits everything and exits the game
def quit_all():
    cv2.destroyAllWindows()
    ut.quit_game()
    keras.clear_session()
    pygame.quit()
```

Figure 11: The 'quit_all' subroutine.

6.2.5 Improving the gesture recognition accuracy

In order to improve the gesture recognition accuracy, the initial method was to use a depth camera. Originally, the game captures the users hand colour and then masks the hand gesture image to make the hand white and the background black. However, under certain lighting conditions and different background colours, the masking doesn't function well. This makes the masked image inaccurate. The figure below presents the sort of results achieved by masking in the best- and worst-case scenario.

Figure 12: The best (on the left) to the worst (on the right) masked gesture images.

From the figure above, the image on the right barely resembles of a hand. Hence the CNN model struggles to identify these forms of images. The masked image can be this bad due to the following reasons:

- User did not scan his/her hand correctly. This results in the wrong hand colour being captured. Hence, the system is masking the image according to a different colour and this results in bad masking.
- The lighting conditions changed after the user scanned their hand. Hence the system is facing the same issue as above.
- Lastly, there may be a background which is similar in colour to the user's hand colour. This results in white spots in different areas which may even be far from the hand. This is likely the cause for the worst masked image shown in figure 12.

Hence, the best way to resolve this issue is by using a depth camera to mask the image. Using a depth camera would eliminate the colour related issues which makes the masking process inaccurate in some cases.

The first depth camera provided by the supervisor was the ZED camera by STEREOLABS. This camera is illustrated in figure 13. Using this camera with the Jetson Nano was quite tough required a lot of troubleshooting in order to make it work.

Figure 13: ZED Depth Camera by STEREOLABS (STEREOLABS).

After successfully making the camera work with the Jetson Nano, it was seen to be inaccurate at capturing depth as shown in figure 14.

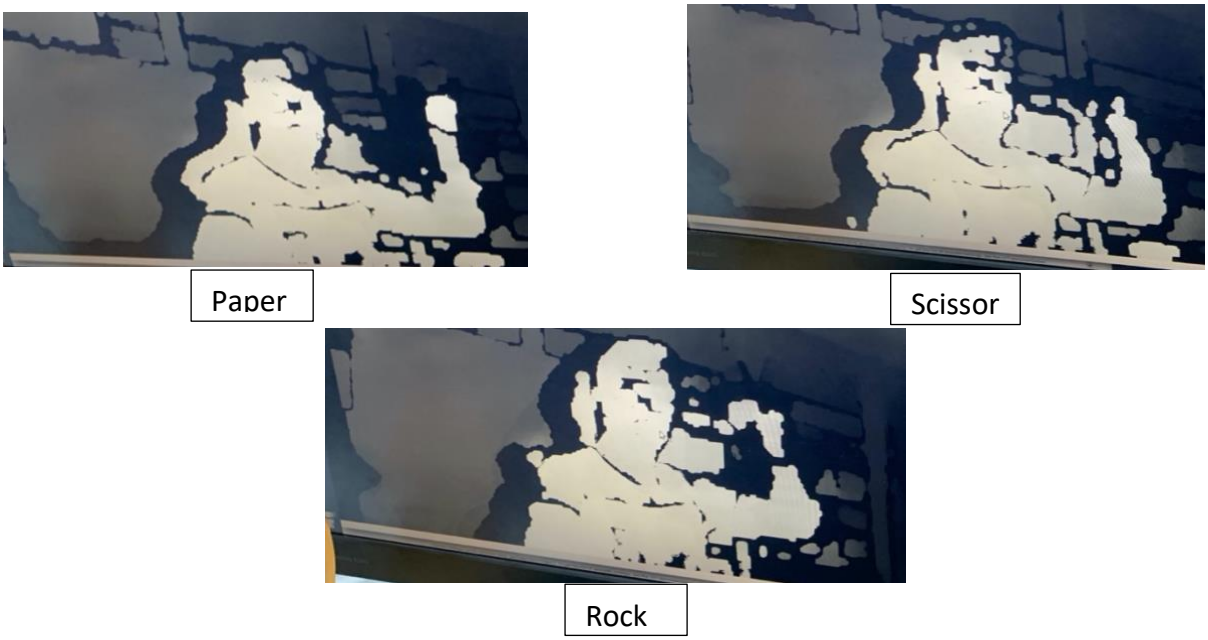


Figure 14: Gesture images captured with ZED depth camera.

From the figure above, the depth images captured are very pixelated and it is very difficult to tell the hand gestures apart. The ZED depth camera isn't very accurate as it uses to cameras and compares the images from both by using the difference in angle within the pixels from each stream. Hence, this depth camera could not be used to improve the gesture recognition.

Next, the supervisor provided another depth camera named Pico Flexx which is manufactured by PmdTec. This camera uses infrared to sense the depth unlike the dual camera setup of the ZED camera. This results in a much better depth image for gesture recognition. Figure 15 presents the Pico Flexx camera.

Figure 15: The Pico Flexx camera by PmdTec (PmdTec).

This camera worked very well for capturing hand gestures which were an accurate representation. However, this camera could not function with the Jetson Nano. The developer kit provided by PmdTec did not support ARM processor on the Jetson Nano. This was discovered, after lots of troubleshooting and contacting PmdTec for support. Due to this problem, this camera could not be used but it would've been a very good solution. The figure below presents the gesture images caught on a Desktop PC with the Pico Flexx camera.

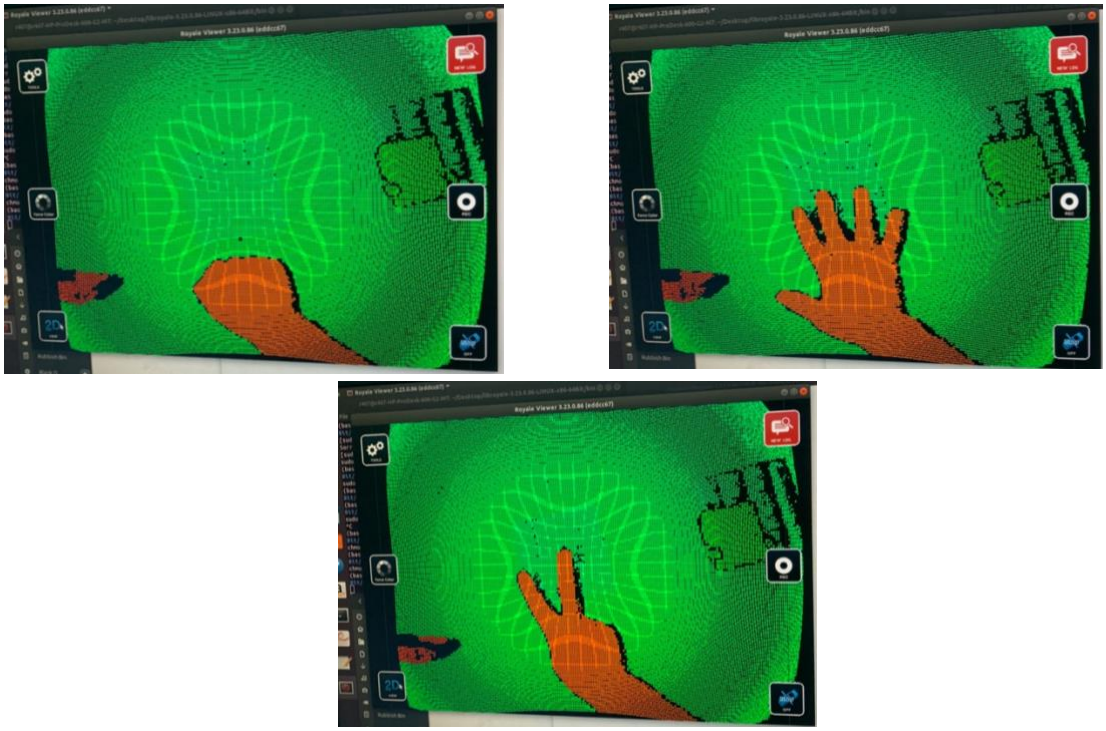


Figure 16: Depth images caught by Pico Flexx camera.

From the figure above, the depth camera would've been very good for capturing the gesture images but wasn't supported by the Jetson Nano ARM processor. Hence, it could not be used. The two depth cameras tired out could not be used due to lack of accuracy or compatibility. The troubleshooting took a lot of time and there were no other depth cameras available. Hence, supervisor suggested to capture more gesture images and retrain the CNN model to improve accuracy.

To do so, the game was setup on another Jetson Nano for the Open House event which has lots of students visiting and trying the game and other projects developed. This allowed for approximately 500 hundred gesture images to be captured. The CNN model was then retrained within Jupyter using these newly added images. The newly trained network was tested within the games and gestures could be recognised in some new angles as well now. This was possible as original user images were capture which will clearly show how users are doing the gestures. Hence, the new data was quite useful for improving the recognition ability.

6.2.6 Miscellaneous work completed

Apart from the improvements made to the game discussed above, there was lots of other work completed such as:

- Installing ROS and all required packages on Jetson Nano.
- Creating the robot design model within ROS so code can be simulated within ROS.
- Installing the motor and motor drivers onto the robot frame.
- Setting up another Jetson Nano with Ubuntu and all packages required to run the game.

- Creating an OpenCV install script for Jetson Nano as installing OpenCV was quite a troublesome procedure. This script can be used by future students if OpenCV installation is required.
- Creating an install script that walks user through installing all necessary packages onto the Jetson Nano. This was done because a lot of time was wasted acquiring the correct versions of packages as certain version are not supported.

6.3 Evaluation of Gantt Chart

A Gantt chart is used to breakdown the project into smaller tasks and organise them by listing the requirements along with the start and end dates for the tasks to be completed. The chart can also use illustrate how a project timeline could be improved via the use of slack to improve cost and time outcomes. Figure 17 illustrates the initial Gantt chart created for this project, within it is a detailed estimated timeline of each task to be completed.

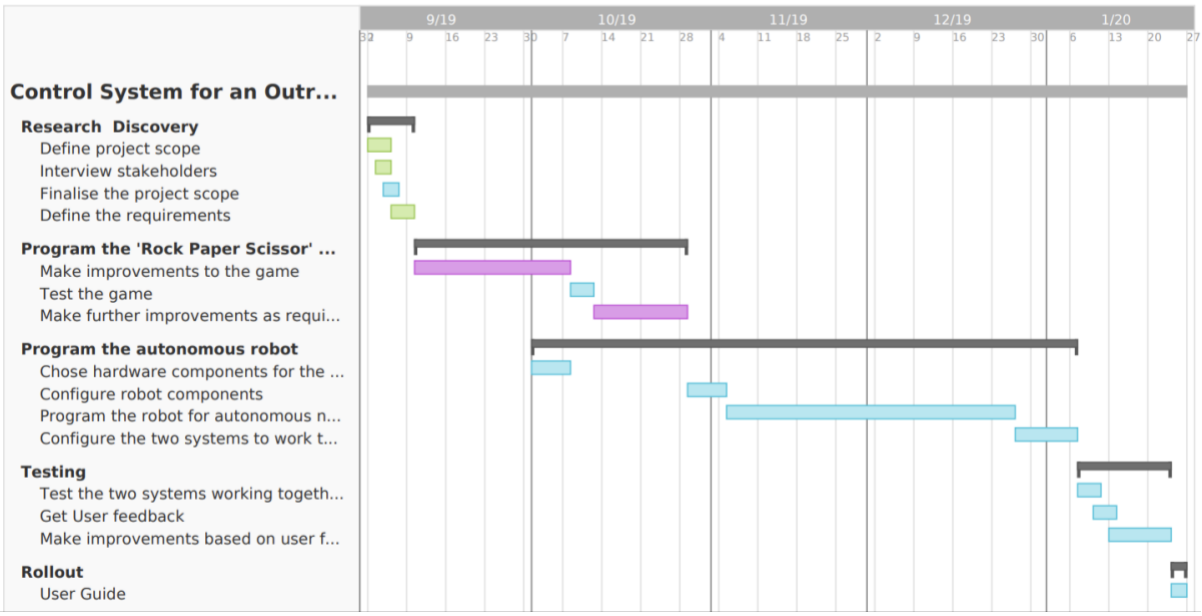


Figure 17: Initial Gantt Chart.

The Gantt chart presented above was created at the beginning of the project and due to the ZLAC706-RC motor driver being faulty in the 12th week, the project scope had to be altered significantly. Hence, due to uncontrollable circumstances the project Gantt chart was revised in week 12 according to the new project objectives. The revised Gantt chart is presented in figure 6 below.

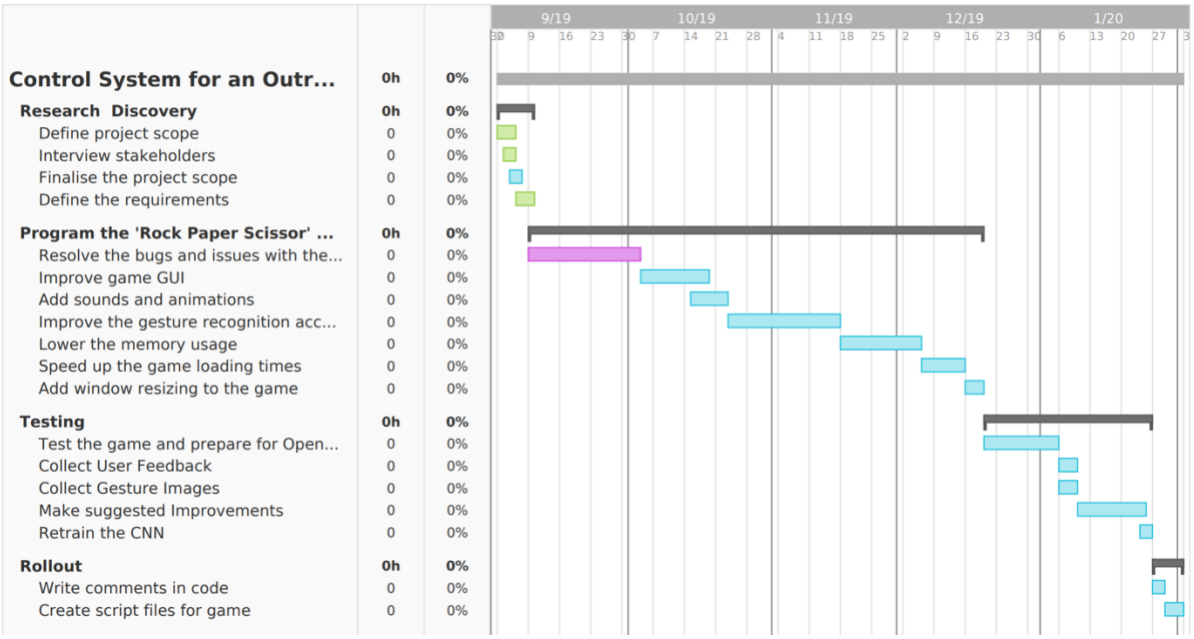


Figure 18: Revised Gantt Chart.

The revised Gantt chart was quite accurate throughout the remainder of the project. After the motor drivers stopped working, the main scope of the project changed to improving the game. Hence, a lot more time was spent on the game and much more significant improvements could be made to the game. The revised Gantt chart was quite accurate for most of the time frames. Some project objectives took longer than planned while others did not take as long as expected. Hence, overall the project could be completed within the 20-week placement.

7. Future Work

To further improve the game, integration of a depth camera is required. While all the bugs and issue with the game were resolved through this project, the gesture recognition accuracy could not be improved significantly. The gesture recognition struggles under some lighting conditions and if the background has objects with the same colour as the user's hand. Unfortunately, the two depth cameras available to use during the project were either inaccurate or incompatible. Hence, they could not be integrated into the game. Once a depth camera is used, the gesture recognition will be close to 100% accurate and will also allow for a smaller sample size for the CNN model.

Programming the robot to drive autonomously and carry the game module is also an area of future work as that is the vision for the entire project. Unfortunately, this component of the project was removed due to faulty parts at a late stage of the project. Hence, it is now a separate project for another student to undertake.

8. Personal Project Reflections

Now that it's over, what are my first thoughts about this overall project? Are they mostly positive or negative?

My thoughts about the project are mostly positive as I believe I made significant improvements and put in a good effort into the work.

What were some of the most interesting discoveries I made while working on this project? About the problem? About myself? About others?

The most interesting aspects was learning how PyGame operates and how to use the functions available within it. At first it was difficult as I had to research each and every subroutine. However, as time passed, I gained a lot of knowledge and was much better at it. I also learned how to install Ubuntu packages and write script files for installation which I had never previously done.

What were some of my most challenging moments and what made them so?

One of the most challenging moments was to read and understand the game code at the beginning of placement. The code was over 1800 lines long and was written in Python and used a lot of the PyGame library functions which I had never seen before. Hence, it took a very long time to understand how the game operates.

Another challenging part was using the motor and motor drivers initially. This is because the manual as well as the provided software were in Chinese. This meant I had to work with my partner to understand how to operate the two.

What most got in the way of my progress, if anything?

The faulty motor driver effected the project scope largely. Since it happened so late, I had also spent a few weeks working on ROS, all of which went to waste upon the parts being faulty.

Were my milestones and goals mostly met, and how much did I deviate from them if any?

The initial project objectives were not met due to faulty parts. However, upon revising the objectives, all the required tasks were completed apart from the depth camera integration. Hence, there was a significant deviation in week 12 when the objectives were revised due to uncontrollable reasons but apart from that there was no deviation.

What did I learn were my greatest strengths? My biggest areas for improvement?

During the placement, I felt I could adapt to new challenges well as all the work I was doing, I had little experience in. Despite that, I was able to get assistance from my supervisor and the internet to learn how I can complete certain tasks.

9. Conclusion

Based on the revised project scope and results, this project can be deemed successful as all the required improvements were made apart from integrating a depth camera into the game. Some of the project could have been completed quicker if certain softwares/packages did not require troubleshooting. The main drawback of the project was the faulty motor driver for the robot. The issue forced significant changes to the project. Although it could not be completed due to uncontrollable circumstances, the extra time allowed for much more significant improvements to be made to the 'Rock Paper Scissors' game.

The game now functions perfectly with no issues or bugs. It can be run on the Jetson Nano with ease, since the memory consumption has been lowered significantly. The game was showcased at the Open House event, which was held in January. The visitors were pleased with the game experience and enjoyed it. The supervisor at Nanyang Polytechnic is pleased with the improvements made to the game, the only improvement required is to replace the webcam with a depth camera and the game will become very accurate at gesture recognition.

Overall the project was a success for Nanyang Polytechnic and was also a great learning experience for me. Working on this project lead to me developing a lot of engineering skills such as programming in Python, using PyGame libraries, training CNN models as well as working individually with little support. The project work also helped me improve project-based learning, time management and communication with stakeholders.

10. References

Express, A. ZLTECH - ZLAC706-RC. Retrieved from https://www.aliexpress.com/item/32971754213.html

PmdTec. Pico Flexx. Retrieved from https://pmdtec.com/picofamily/

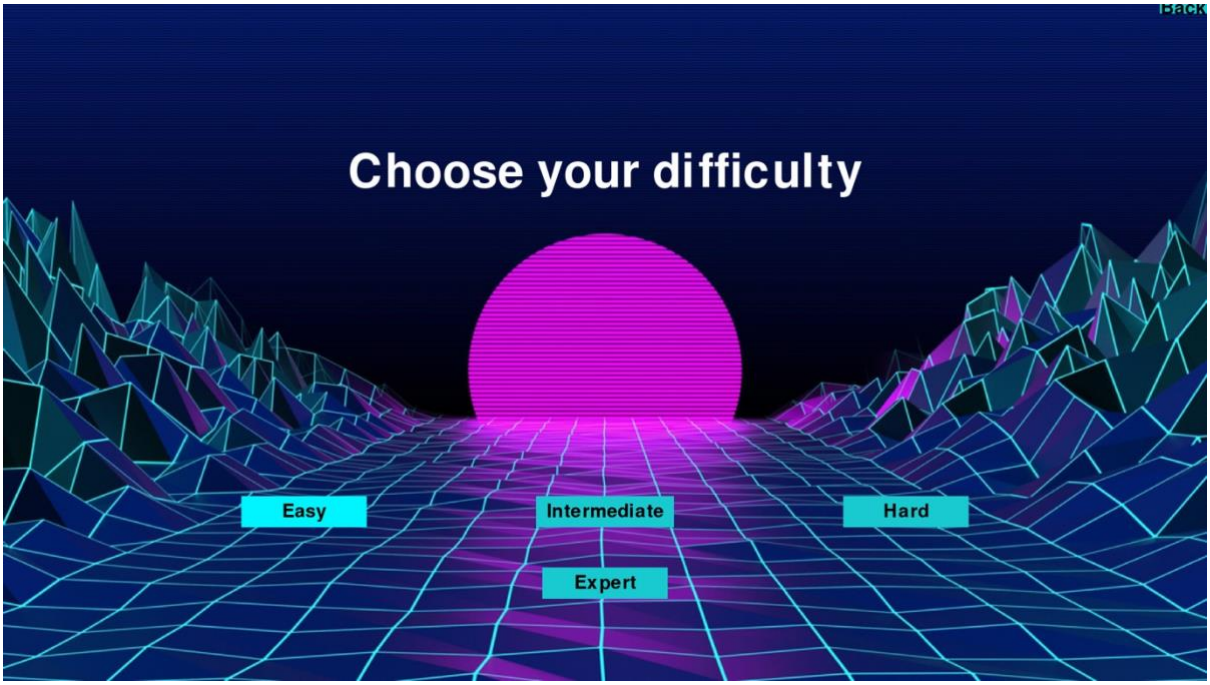
Polytechnic, N. Nanyang Polytechnic: The Innovative Polytechnic. Retrieved from https://www.nyp.edu.sg/schools.html

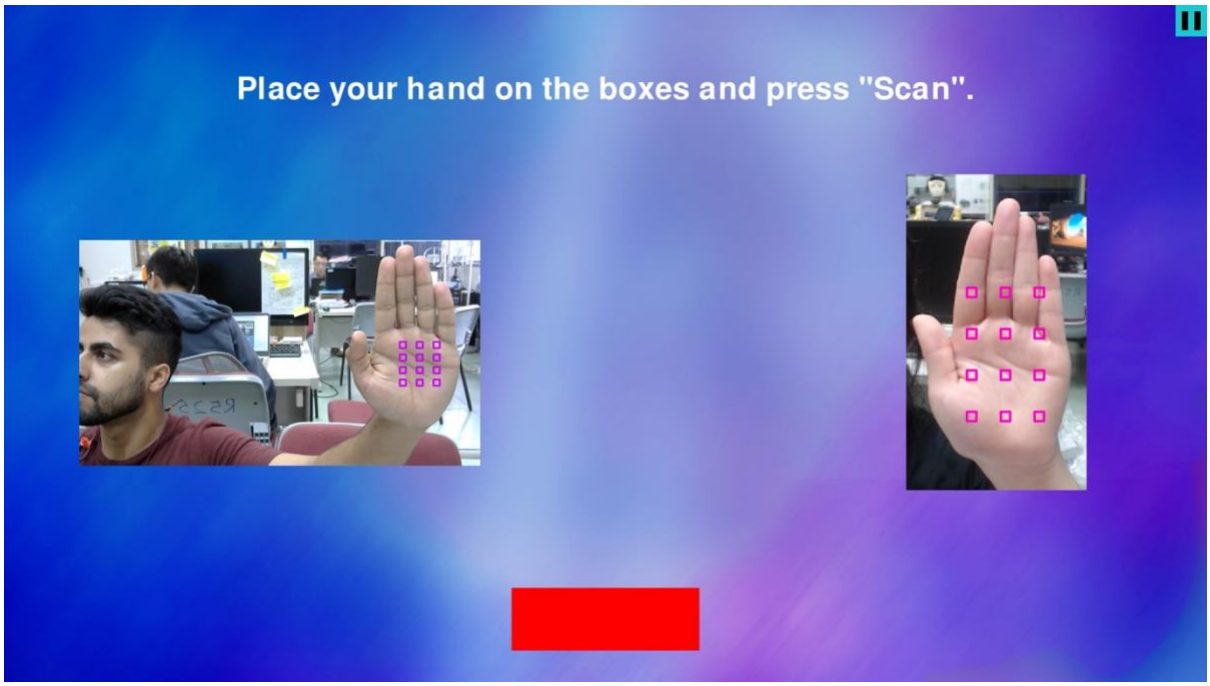
Pygame.org. Pygame: Pygame Documentation. Retrieved from https://www.pygame.org/docs/index.html

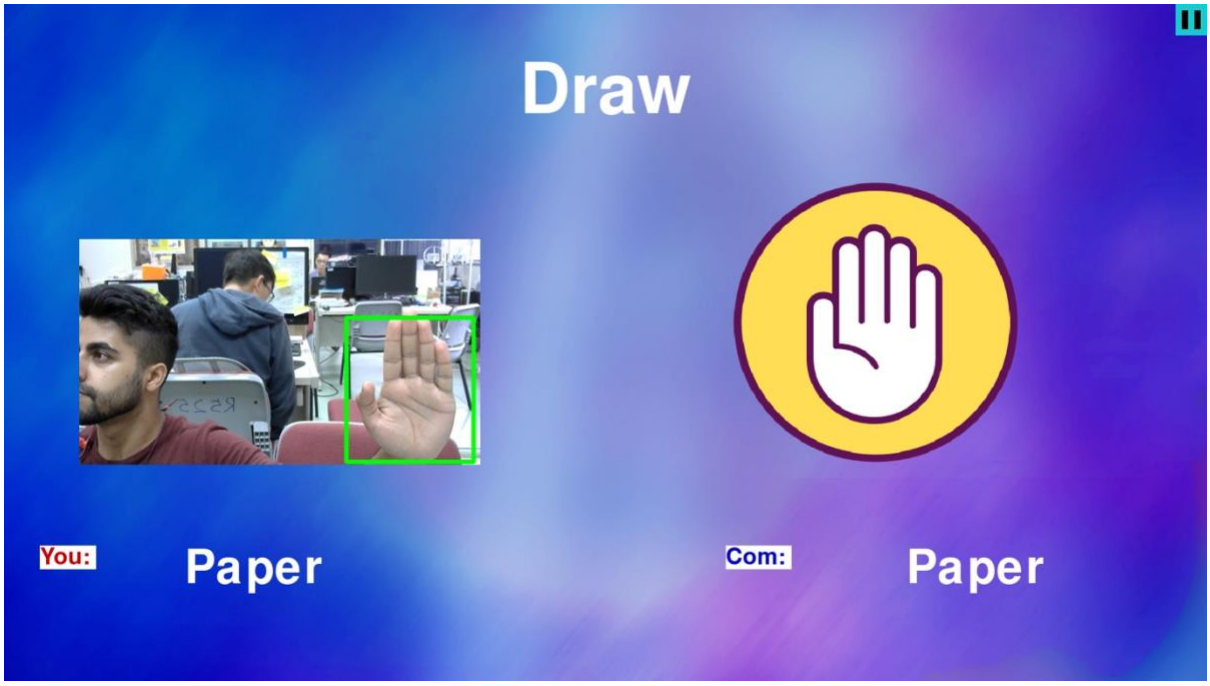
STEREOLABS. ZED Depth Camera. Retrieved from https://www.stereolabs.com/zed/

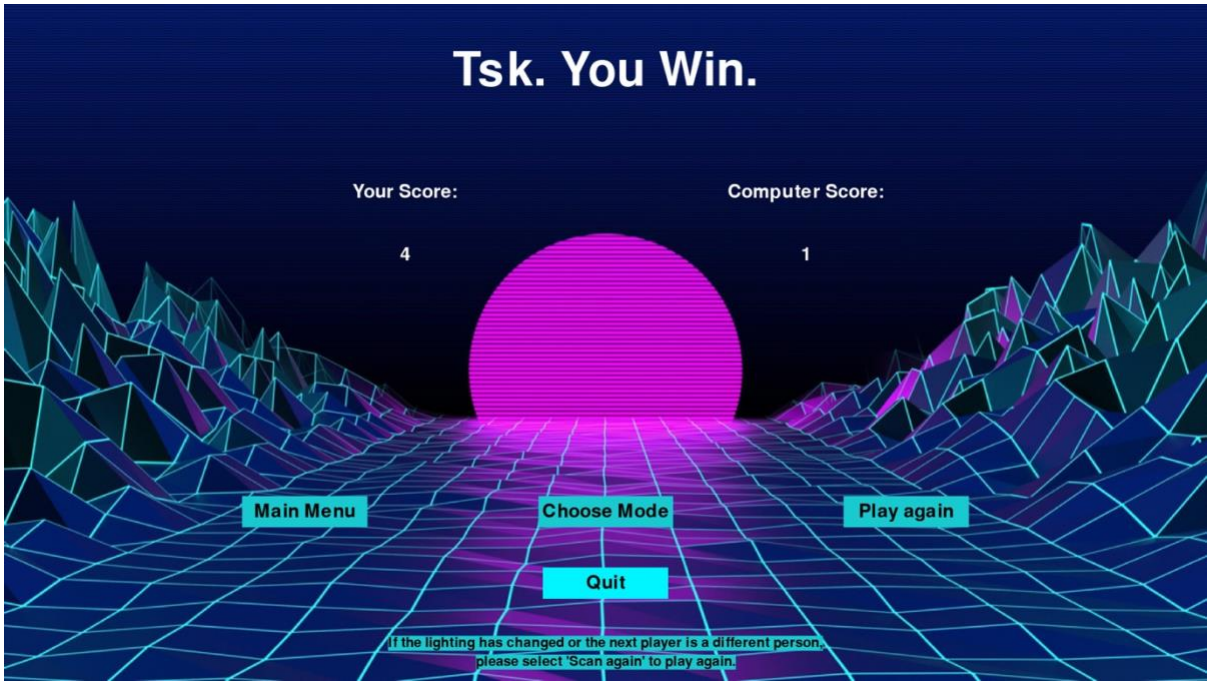
Appendix A

New Game GUI

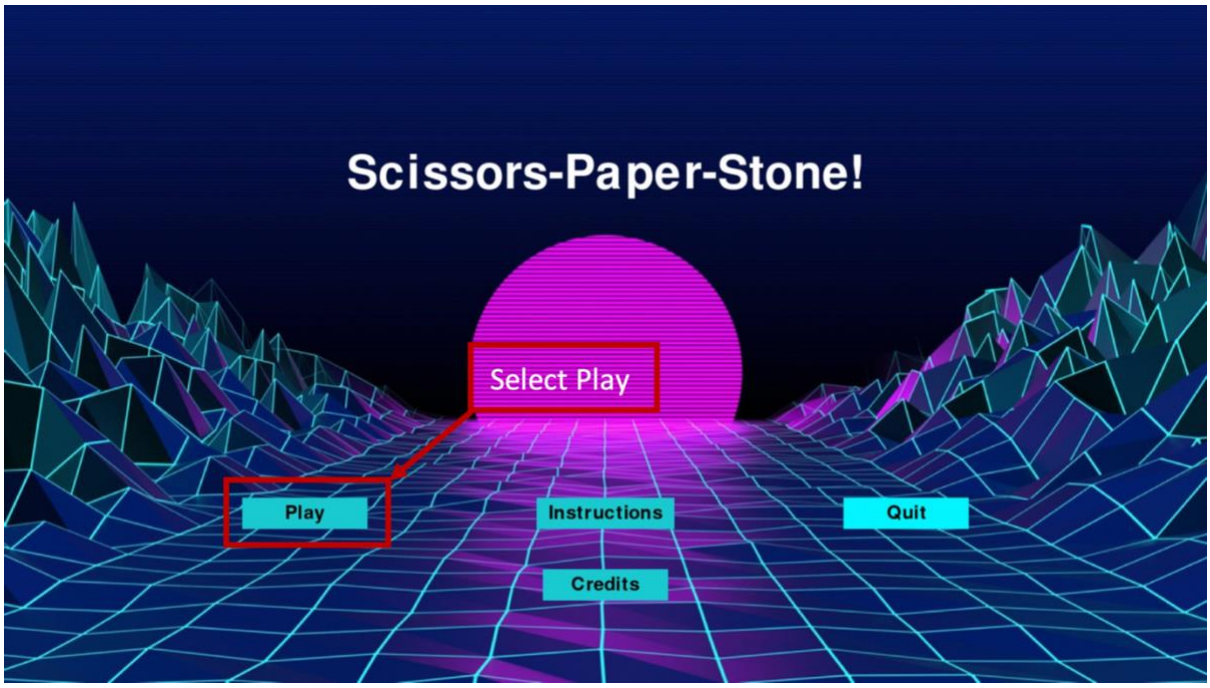


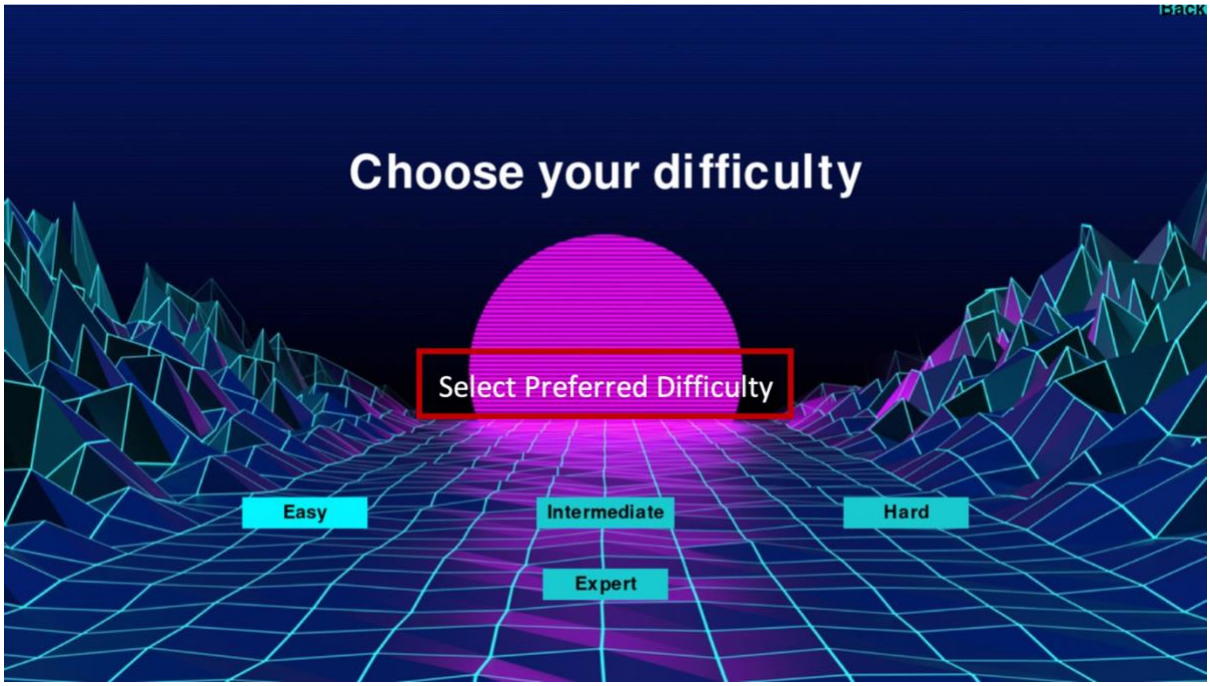

Place your hand on the boxes and press "Scan".

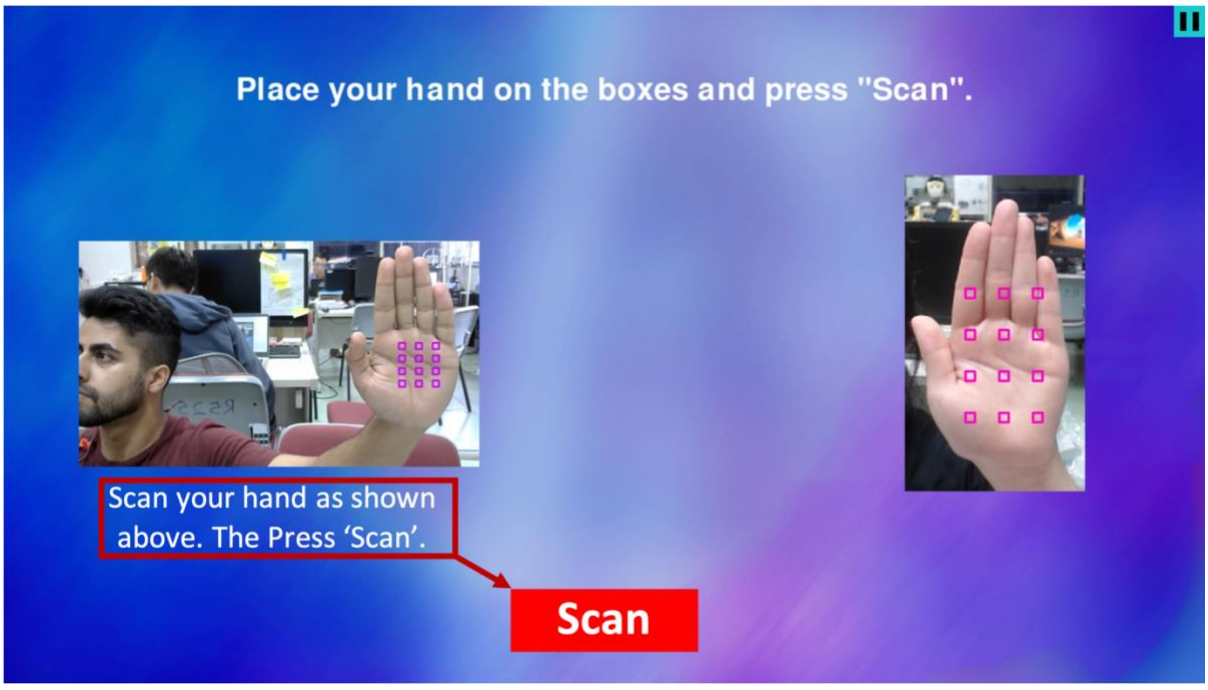

Draw
You:
Paper
Com:
Paper

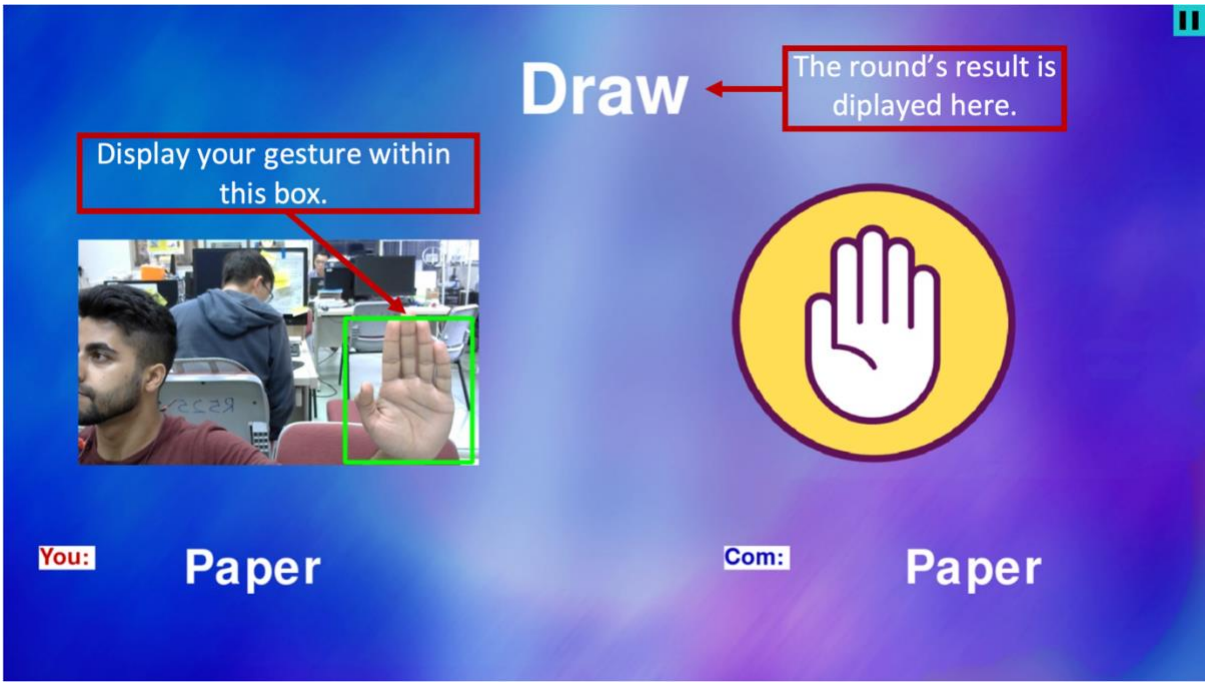


Game Instruction Screens




Choose your difficulty
Select Preferred Difficulty
Easy
Intermediate
Hard
Expert


Place your hand on the boxes and press "Scan".
Scan your hand as shown above. The Press 'Scan'.
Scan



Install OpenCV script

```
# ------------------------------------------------------------------ |
OPENCV_VERSION='4.1.1'      # Version to be installed
OPENCV_CONTRIB='NO'         # Install OpenCV's extra modules (YES/NO)
# ------------------------------------------------------------------ |

# 1. KEEP UBUNTU OR DEBIAN UP TO DATE
```

```
sudo apt-get -y update
# sudo apt-get -y upgrade       # Uncomment to install new versions of packages currently installed
# sudo apt-get -y dist-upgrade  # Uncomment to handle changing dependencies with new vers. of pack.
# sudo apt-get -y autoremove    # Uncomment to remove packages that are now no longer needed


# 2. INSTALL THE DEPENDENCIES

sudo apt-get install -y build-essential cmake

sudo apt-get install -y qt5-default libvtk6-dev

sudo apt-get install -y zlib1g-dev libjpeg-dev libwebp-dev libpng-dev libtiff5-dev libjasper-dev \
                libopenexr-dev libgdal-dev

sudo apt-get install -y libdc1394-22-dev libavcodec-dev libavformat-dev libswscale-dev \
                libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev yasm \
                libopencore-amrnb-dev libopencore-amrwb-dev libv4l-dev libxine2-dev

sudo apt-get install -y libtbb-dev libeigen3-dev

sudo apt-get install -y python-dev  python-tk  pylint  python-numpy  \
                python3-dev python3-tk pylint3 python3-numpy flake8


sudo apt-get install -y ant default-jdk

wget https://github.com/opencv/opencv/archive/${OPENCV_VERSION}.zip
unzip ${OPENCV_VERSION}.zip && rm ${OPENCV_VERSION}.zip
mv opencv-${OPENCV_VERSION} OpenCV

if [ $OPENCV_CONTRIB = 'YES' ]; then
  wget https://github.com/opencv/opencv_contrib/archive/${OPENCV_VERSION}.zip
  unzip ${OPENCV_VERSION}.zip && rm ${OPENCV_VERSION}.zip
  mv opencv_contrib-${OPENCV_VERSION} opencv_contrib
  mv opencv_contrib OpenCV
fi

cd OpenCV && mkdir build && cd build

if [ $OPENCV_CONTRIB = 'NO' ]; then
cmake -DWITH_QT=ON -DWITH_OPENGL=ON -DFORCE_VTK=ON -DWITH_TBB=ON -DWITH_GDAL=ON \
```

```
    -DWITH_XINE=ON -DENABLE_PRECOMPILED_HEADERS=OFF ..
fi

if [ $OPENCV_CONTRIB = 'YES' ]; then
cmake -DWITH_QT=ON -DWITH_OPENGL=ON -DFORCE_VTK=ON -DWITH_TBB=ON -
DWITH_GDAL=ON \
    -DWITH_XINE=ON -DENABLE_PRECOMPILED_HEADERS=OFF \
    -DOPENCV_EXTRA_MODULES_PATH=../opencv_contrib/modules ..
fi

make -j4
sudo make install
sudo ldconfig
```

Install Jetson Nano packages script

Install pip:
sudo apt install python3-pip

Install python3:
sudo apt-get install python3.7-dev

Install TensorFlow for Jetson Nano:
https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html

sudo apt-get update
sudo apt-get install libhdf5-serial-dev hdf5-tools libhdf5-dev zlib1g-dev zip libjpeg8-dev

sudo apt-get install python3-pip
sudo pip3 install -U pip
sudo pip3 install -U numpy==1.16.1 future==0.17.1 mock==3.0.5 h5py==2.9.0
keras_preprocessing==1.0.5 keras_applications==1.0.8 gast==0.2.2 enum34 futures
testresources setuptools protobuf
sudo pip3 install --pre --extra-index-url
https://developer.download.nvidia.com/compute/redist/jp/v42 tensorflow-gpu

Install Keras:
https://github.com/hsekia/learning-keras/wiki/How-to-install-Keras-to-Ubuntu-18.04

Install Pygame:
sudo apt-get install mercurial
hg clone https://bitbucket.org/pygame/pygame
cd pygame

sudo apt-get install libsdl-dev libsdl-image1.2-dev libsdl-mixer1.2-dev libsdl-ttf2.0-dev
sudo apt-get install libsmpeg-dev libportmidi-dev libavformat-dev libswscale-dev

sudo apt-get install python3-dev python3-numpy

python3 setup.py build
sudo python3 setup.py install

Install PygameMenu:
pip install pygame-menu

Appendix B

Game Code

```
#!/usr/bin/env python3
import pygame
import time
import cv2
import numpy as np
import time
import keras
import os
import random
import tensorflow as tf
import time
import pygameMenu
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
from pygame.locals import *

global cap
global camera_status
global display_width, display_height, text_size, button_w, button_h, text_size_1
global gameDifficulty
matches = 0
init_sound = 0
######################################################################
import random
X = '\033[0m'
Bold = '\033[1;36m'
HighB = '\033[1;44m'

winEas = loseEas = tieEas = winInt = loseInt = tieInt = winHard = loseHard = tieHard = winExp = loseExp = tieExp = winspec =
losespec = tiespec = 0.0

hiddenfound = False

buildTMatrix = {'rr': 1, 'rp': 1, 'rs': 1, 'pr': 1, 'pp': 1, 'ps': 1, 'sr': 1, 'sp': 1, 'ss': 1}
buildTMatrixL = {'rr': 1, 'rp': 1, 'rs': 1, 'pr': 1, 'pp': 1, 'ps': 1, 'sr': 1, 'sp': 1, 'ss': 1}
buildTMatrixT = {'rr': 1, 'rp': 1, 'rs': 1, 'pr': 1, 'pp': 1, 'ps': 1, 'sr': 1, 'sp': 1, 'ss': 1}

n = 3
m = 3
tMatrix = [[0] * m for i in range(n)]
tMatrixL = [[0] * m for i in range(n)]
tMatrixT = [[0] * m for i in range(n)]

probabilitiesRPS = [1/3,1/3,1/3]

buildTMatrixrpsclsp = {'rr': 1, 'rp': 1, 'rsc': 1, 'rl': 1, 'rsp': 1, 'pr': 1, 'pp': 1, 'psc': 1, 'pl': 1, 'psp': 1, 'scr': 1, 'scp': 1, 'scsc': 1, 'scl': 1,
'scsp': 1, 'lr': 1, 'lp': 1, 'lsc': 1, 'll': 1, 'lsp': 1, 'spr': 1, 'spp': 1, 'spsc': 1, 'spl': 1, 'spsp': 1}
buildTMatrixLrpsclsp = {'rr': 1, 'rp': 1, 'rsc': 1, 'rl': 1, 'rsp': 1, 'pr': 1, 'pp': 1, 'psc': 1, 'pl': 1, 'psp': 1, 'scr': 1, 'scp': 1, 'scsc': 1, 'scl':
1, 'scsp': 1, 'lr': 1, 'lp': 1, 'lsc': 1, 'll': 1, 'lsp': 1, 'spr': 1, 'spp': 1, 'spsc': 1, 'spl': 1, 'spsp': 1}
buildTMatrixTrpsclsp = {'rr': 1, 'rp': 1, 'rsc': 1, 'rl': 1, 'rsp': 1, 'pr': 1, 'pp': 1, 'psc': 1, 'pl': 1, 'psp': 1, 'scr': 1, 'scp': 1, 'scsc': 1, 'scl':
1, 'scsp': 1, 'lr': 1, 'lp': 1, 'lsc': 1, 'll': 1, 'lsp': 1, 'spr': 1, 'spp': 1, 'spsc': 1, 'spl': 1, 'spsp': 1}

sheldon = 5
cooper = 5
tMatrixrpsclsp = [[0] * sheldon for i in range(cooper)]
tMatrixLrpsclsp = [[0] * sheldon for i in range(cooper)]
tMatrixTrpsclsp = [[0] * sheldon for i in range(cooper)]

probabilitiesrpsclsp = [1/5,1/5,1/5,1/5,1/5]
######################################################################
# Load CNN model for predicting gestures
'''
```

```
from keras.backend.tensorflow_backend import set_session
config = tf.ConfigProto()
config.gpu_options.allow_growth = True  # dynamically grow the memory used on the GPU
config.log_device_placement = True  # to log device placement (on which device the operation ran)
sess = tf.Session(config=config)
set_session(sess)  # set this TensorFlow session as the default session for Keras
'''
model = keras.models.load_model('CNN/binary_weight.h5')
#interpreter = tf.lite.Interpreter("CNN/converted_model.tflite")
#interpreter.allocate_tensors()
#input_details = interpreter.get_input_details()
#output_details = interpreter.get_output_details()
#model = converted_model.tflite
######################################################################

directory ="/home/nyp/Desktop/images"

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

######################################################################


######################################################################


gameDisplay = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
pygame.display.set_caption('Scissors-Paper-Stone')
clock = pygame.time.Clock()

# Setting window size, caption, icon and clock for FPS
display_width = gameDisplay.get_width()  #1200 need to give these screen size value if using window resizing
display_height = gameDisplay.get_height()  #700
text_size = 30
button_w = 200
button_h = 50
text_size_1 = 75
#uncomment the 'gameDisplay' line below and all 'resizeDisplay' to have a resizeable display. Also Comment out the line
'gameDisplay' line above.
#gameDisplay = pygame.display.set_mode((display_width, display_height), HWSURFACE|DOUBLEBUF|RESIZABLE)

def resizeDisplay():
    global display_width, display_height, button_w, button_h, text_size, text_size_1
    pygame.event.pump()
    event=pygame.event.wait()
    if event.type==QUIT: quit_all()
    elif event.type==VIDEORESIZE:
        display_width = event.w
        display_height = event.h
        button_w = int(200*(display_width/1200))
        button_h = int(50*(display_height/700))
        text_size = int(30*((display_width+display_height)/1900))
        text_size_1 = int(75*((display_width+display_height)/1900))
        gameDisplay = pygame.display.set_mode((display_width, display_height), HWSURFACE|DOUBLEBUF|RESIZABLE)
        pygame.display.flip()
        return display_width, display_height

######################################################################
# Checking if camera is plugged in
def check_camera():
    global cap
    count = 0
    while True:
        cap = cv2.VideoCapture(0)
        cap.set(3, 1280)
```

```
        cap.set(4, 720)
        if not cap.isOpened():
            gameDisplay.fill(c.black)
            text('Please plug in a camera to play the game', 50, c.white, display_width, display_height, True)
            ut.update_screen(clock)
            count = 1
            ut.quit_x()
        else:
            if count == 1:
                camera_detected()
                break
            else:
                break
    #cap.release()
    intro()

def camera_detected():
    start = time.time()
    while time.time() - start < 4:
        gameDisplay.fill(c.black)
        text('Camera detected! You will be directed to the main menu.', 50, c.white, display_width, display_height, True)
        ut.update_screen(clock)
        ut.quit_x()

######################################################################
# 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'''


    textSurface = font.render(msg, True, color, back_color)
    return textSurface, textSurface.get_rect() #return text surface object and (get)rectangular surface

def text(msg, text_size, color, x, y, center_of_box = False, back_color=None, center_point=False, font_type='freesansbold.ttf'):
    # create font object
    font = pygame.font.Font(font_type, text_size)

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

    if center_of_box:
        textRect.center = (x // 2, y // 2)
    elif center_point:
        textRect.center = (x, y)
    else:
        textRect = (x, y)

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

######################################################################
# Button
def button_text(msg, text_size, color, font_type, x, y, w, h):
    # create font object
    font = pygame.font.Font(font_type, text_size)

    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 rectangle to be at the designated place

    gameDisplay.blit(textSurf, textRect) # draw image, in this case 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'):
    global pressed
    global matches
```

```
global gameDifficulty


mouse = pygame.mouse.get_pos()
#print(mouse)
click = pygame.mouse.get_pressed()
#print(click)

# coordinate of button adjusting
if placement == 'center':
    x = x - w//2
    y = y - h//2

elif placement == 'top right':
    x = x - w

elif placement == 'bottom left':
    y = y - h

elif placement == 'bottom right':
    x = x - w
    y = y - h

pygame.event.pump()
if x < mouse[0] < x+w and y < mouse[1] < y+h: # if mouse inside button
    pygame.draw.rect(gameDisplay, light_up_color, (x, y, w, h))
    pygame.event.pump()
    #pygame.event.wait()
    if click[0] == 1:
    #for event in pygame.event.get():
        if action: # if there is action, do the action
            if action == 'paused':
                global resume
                pygame.mixer.music.set_volume(0.3)
                sound.pause()
                pause_menu()
            else:
                pygame.mixer.music.set_volume(1.0)
                if action == 'game_1':
                    if scanned == 1:
                        sound.play()
                        game_screen()
                    else:
                        pause_scan_first()
                elif action == 'intro':
                    sound.select()
                    intro()
                elif action == 'play':
                    sound.select()
                    chooseMode()
                elif action == 'quit':
                    quit_all()
                elif action == 'easy':
                    sound.select()
                    gameDifficulty = 'easy'
                    hand_scan_screen()
                elif action == 'intermediate':
                    sound.select()
                    gameDifficulty = 'intermediate'
                    hand_scan_screen()
                elif action == 'hard':
                    sound.select()
                    gameDifficulty = 'hard'
                    hand_scan_screen()
                elif action == 'expert':
                    sound.select()
```

```
                gameDifficulty = 'expert'
                hand_scan_screen()
            elif action == 'back':
                sound.select()
                intro()
            elif action == 'hand_scan':
                sound.play()
                pressed = 1
                return pressed
            elif action == 'scan_screen':
                sound.select()
                hand_scan_screen()
            elif action == 'tutt_1':
                if previous  == 'intro' or previous == 'paused':
                    music.stop()
                    music.instruc()
                    sound.select()
                    tutorial_one()
                else:
                    sound.select()
                    tutorial_one()
            elif action == 'tutt_2':
                sound.select()
                tutorial_two()
            elif action == 'tutt_3':
                sound.select()
                tutorial_three()
            elif action == 'tutt_4':
                sound.select()
                tutorial_four()
            elif action == 'tutt_5':
                sound.select()
                tutorial_five()
            elif action == 'credits_one':
                if previous  == 'intro':
                    music.stop()
                    music.credits()
                    sound.select()
                    credits_one()
                else:
                    sound.select()
                    credits_one()
            elif action == 'credits_two':
                sound.select()
                credits_two()
            elif action == 'credits_three':
                sound.select()
                credits_three()
    else:
        pygame.draw.rect(gameDisplay, default_color, (x, y, w, h))

    if msg == '':
        pass
    else:
        button_text(msg, text_size, text_color, font_type, x, y, w, h)

##################################################################
def tutorial_one():
    while True:
        global display_width, display_height, button_w, button_h, text_size, text_size_1
        gameDisplay.blit(pygame.transform.scale(pic.tutorial_1, (display_width, display_height)), (0, 0))
        button('bottom left', 30, display_height - 20, button_w, button_h, c.red_dim, c.red_lit, 'Main Menu', text_size, c.white,
'tutt_1', 'intro')
        button('bottom right', display_width - 30, display_height - 20, button_w, button_h, c.red_dim, c.red_lit, 'Next', text_size,
c.white, 'tutt_1', 'tutt_2')
        ut.update_screen(clock)
```

```
        ut.quit_x()
def tutorial_two():
    while True:
        global display_width, display_height, button_w, button_h, text_size, text_size_1
        gameDisplay.blit(pygame.transform.scale(pic.tutorial_2, (display_width, display_height)), (0, 0))
        button('bottom left', 30, display_height - 20, button_w, button_h, c.red_dim, c.red_lit, 'Back', text_size, c.white, 'tutt_2',
'tutt_1')
        button('bottom right', display_width - 30, display_height - 20, button_w, button_h, c.red_dim, c.red_lit, 'Next', text_size,
c.white, 'tutt_2', 'tutt_3')
        ut.update_screen(clock)
        ut.quit_x()
def tutorial_three():
    while True:
        global display_width, display_height, button_w, button_h, text_size, text_size_1
        gameDisplay.blit(pygame.transform.scale(pic.tutorial_3, (display_width, display_height)), (0, 0))
        button('bottom left', 30, display_height - 20, button_w, button_h, c.red_dim, c.red_lit, 'Back', text_size, c.white, 'tutt_3',
'tutt_2')
        button('bottom right', display_width - 30, display_height - 20, button_w, button_h, c.red_dim, c.red_lit, 'Next', text_size,
c.white, 'tutt_3', 'tutt_4')
        ut.update_screen(clock)
        ut.quit_x()
def tutorial_four():
    while True:
        global display_width, display_height, button_w, button_h, text_size, text_size_1
        gameDisplay.blit(pygame.transform.scale(pic.tutorial_4, (display_width, display_height)), (0, 0))
        button('bottom left', 30, display_height - 20, button_w, button_h, c.red_dim, c.red_lit, 'Back', text_size, c.white, 'tutt_4',
'tutt_3')
        button('bottom right', display_width - 30, display_height - 20, button_w, button_h, c.red_dim, c.red_lit, 'Next', text_size,
c.white, 'tutt_4', 'tutt_5')
        ut.update_screen(clock)
        ut.quit_x()
def tutorial_five():
    while True:
        global display_width, display_height, button_w, button_h, text_size, text_size_1
        gameDisplay.blit(pygame.transform.scale(pic.tutorial_5, (display_width, display_height)), (0, 0))
        button('bottom left', 30, display_height - 20, button_w, button_h, c.red_dim, c.red_lit, 'Back', text_size, c.white, 'tutt_5',
'tutt_4')
        button('bottom right', display_width - 30, display_height - 20, button_w + 150, button_h, c.red_dim, c.red_lit, 'Exit
Instructions', text_size, c.white, 'tutt_5', 'intro')
        ut.update_screen(clock)
        ut.quit_x()

#################################################################

def intro():
    global display_width, display_height, button_w, button_h, text_size, text_size_1
    music.stop()
    music.intro()
    while True:
        #resizeDisplay()
        gameDisplay.blit(pygame.transform.scale(pic.background, (display_width, display_height)), (0, 0))
        text('Scissors-Paper-Stone!', text_size_1, c.white, display_width, display_height//2*1, True)
        button('center', display_width//4, display_height//4*3, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Play',
text_size, c.black, 'intro', 'play')
        button('center', display_width//4 * 2, display_height//4*3, button_w+20, button_h, c.light_blue_dim, c.light_blue_lit,
'Instructions', text_size, c.black, 'intro', 'tutt_1')
        button('center', display_width//4 * 3, display_height//4*3, button_w, button_h,c.light_blue_dim, c.light_blue_lit, 'Quit',
text_size, c.black, 'intro', 'quit')
        button('center', display_width // 4 * 2, display_height //7*6, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Credits',
text_size, c.black, 'intro', 'credits_one')
        #pygame.display.flip()
        ut.update_screen(clock)
        ut.quit_x()

def chooseMode():
    global display_width, display_height, button_w, button_h, text_size, text_size_1
```

```
    music.stop()
    music.intro()
    while True:
        #resizeDisplay()
        gameDisplay.blit(pygame.transform.scale(pic.background, (display_width, display_height)), (0, 0))
        text('Choose your difficulty', text_size_1, c.white, display_width, display_height//2*1, True)
        button('center', display_width//4, display_height//4*3, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Easy',
text_size, c.black, 'chooseMode', 'easy')
        button('center', display_width//4 * 2, display_height//4*3, button_w+20, button_h, c.light_blue_dim, c.light_blue_lit,
'Intermediate', text_size, c.black, 'chooseMode', 'intermediate')
        button('center', display_width//4 * 3, display_height//4*3, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Hard',
text_size, c.black, 'chooseMode', 'hard')
        button('center', display_width//4 * 2, display_height//7*6, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Expert',
text_size, c.black, 'chooseMode', 'expert')
        button('top right', display_width, 0, button_w-125, button_h-35, c.light_blue_dim, c.light_blue_lit, 'Back', text_size, c.black,
'chooseMode', 'back')
        ut.update_screen(clock)
        ut.quit_x()

def credits_one():
    global display_width, display_height, button_w, button_h, text_size, text_size_1
    while True:
        #resizeDisplay()
        gameDisplay.blit(pygame.transform.scale(pic.credits_1, (display_width, display_height)), (0, 0))
        button('top left', 30, 30, 300, 100, c.light_blue_dim, c.light_blue_lit, 'Main Menu', text_size, c.black, 'credits_one', 'intro')
        button('top right', display_width - 30, 30, 300, 100, c.light_blue_dim, c.green_lit, 'Next', text_size, c.black, 'credits_one',
'credits_two')
        ut.update_screen(clock)
        ut.quit_x()

def credits_two():
    global display_width, display_height, button_w, button_h, text_size, text_size_1
    while True:
        #resizeDisplay()
        gameDisplay.blit(pygame.transform.scale(pic.credits_2, (display_width, display_height)), (0, 0))
        button('top left', 30, 30, 300, 100, c.light_blue_dim, c.light_blue_lit, 'Back', text_size, c.black, 'credits_two', 'credits_one')
        button('top right', display_width - 30, 30, 300, 100, c.light_blue_dim, c.light_blue_lit, 'Next', text_size, c.black,
'credits_two', 'credits_three')
        ut.update_screen(clock)
        ut.quit_x()

def credits_three():
    global display_width, display_height, button_w, button_h, text_size, text_size_1
    while True:
        #resizeDisplay()
        gameDisplay.blit(pygame.transform.scale(pic.credits_3, (display_width, display_height)), (0, 0))
        button('top left', 30, 30, 300, 100, c.light_blue_dim, c.light_blue_lit, 'Back', text_size, c.black, 'credits_three', 'credits_two')
        button('top right', display_width - 30, 30, 300, 100, c.light_blue_dim, c.light_blue_lit, 'Main Menu', text_size, c.black,
'credits_three', 'intro')
        ut.update_screen(clock)
        ut.quit_x()

def SPS_timing(time):
    global display_width, display_height, button_w, button_h, text_size, text_size_1
    b_circle1_w, b_circle1_h = display_width // 3, int(display_height * (3 / 16))
    b_circle2_w, b_circle2_h = display_width // 2, int(display_height * (3 / 16))
    b_circle3_w, b_circle3_h = int(display_width * (2 / 3)), int(display_height * (3 / 16))
    s_circle1_w, s_circle1_h = int(display_width * (5 / 12)), int(display_height * (3 / 16))
    s_circle2_w, s_circle2_h = int(display_width * (7 / 12)), int(display_height * (3 / 16))

    #s_circle3_w, s_circle3_h = int(display_width * (9 / 12)), int(display_height * (3 / 16))
    lit = c.orange
    dim = c.pink_red
    ready_time = 2
    detect_now = False
    global init_sound
```

```
if time < ready_time:
    init_sound = 0
    text('Ready?', 100, c.white, display_width, display_height // 8, True)
    pygame.draw.circle(gameDisplay, dim, (b_circle1_w, b_circle1_h), 50)
    pygame.draw.circle(gameDisplay, dim, (s_circle1_w, s_circle1_h), 30)
    pygame.draw.circle(gameDisplay, dim, (b_circle2_w, b_circle2_h), 50)
    pygame.draw.circle(gameDisplay, dim, (s_circle2_w, s_circle2_h), 30)
    pygame.draw.circle(gameDisplay, dim, (b_circle3_w, b_circle3_h), 50)
    #pygame.draw.circle(gameDisplay, dim, (s_circle3_w, s_circle3_h), 1)
    text('3', 100, c.black, b_circle1_w, b_circle1_h+5, False, None, True)
    text('2', 100, c.black, b_circle2_w, b_circle2_h + 5, False, None, True)
    text('1', 100, c.black, b_circle3_w, b_circle3_h + 5, False, None, True)

elif ready_time < time < ready_time + 0.5:
    init_sound +=1
    text('Scissors...', 100, c.white, display_width, display_height // 8, True)
    pygame.draw.circle(gameDisplay, lit, (b_circle1_w, b_circle1_h), 50)
    pygame.draw.circle(gameDisplay, dim, (s_circle1_w, s_circle1_h), 30)
    pygame.draw.circle(gameDisplay, dim, (b_circle2_w, b_circle2_h), 50)
    pygame.draw.circle(gameDisplay, dim, (s_circle2_w, s_circle2_h), 30)
    pygame.draw.circle(gameDisplay, dim, (b_circle3_w, b_circle3_h), 50)
    #pygame.draw.circle(gameDisplay, dim, (s_circle3_w, s_circle3_h), 1)
    text('3', 100, c.black, b_circle1_w, b_circle1_h+5, False, None, True)
    text('2', 100, c.black, b_circle2_w, b_circle2_h + 5, False, None, True)
    text('1', 100, c.black, b_circle3_w, b_circle3_h + 5, False, None, True)
    gameDisplay.blit(pic.scissors, (display_width*0.6, display_height*0.25)) #650,160
    if init_sound == 1:
        #sound.timing()
        sound.countdown3()

elif ready_time + 0.5 < time < ready_time + 0.5*2:
    init_sound = 0
    text('Scissors...', 100, c.white, display_width, display_height // 8, True)
    pygame.draw.circle(gameDisplay, lit, (b_circle1_w, b_circle1_h), 50)
    pygame.draw.circle(gameDisplay, lit, (s_circle1_w, s_circle1_h), 30)
    pygame.draw.circle(gameDisplay, dim, (b_circle2_w, b_circle2_h), 50)
    pygame.draw.circle(gameDisplay, dim, (s_circle2_w, s_circle2_h), 30)
    pygame.draw.circle(gameDisplay, dim, (b_circle3_w, b_circle3_h), 50)
    #pygame.draw.circle(gameDisplay, dim, (s_circle3_w, s_circle3_h), 1)
    text('3', 100, c.black, b_circle1_w, b_circle1_h+5, False, None, True)
    text('2', 100, c.black, b_circle2_w, b_circle2_h + 5, False, None, True)
    text('1', 100, c.black, b_circle3_w, b_circle3_h + 5, False, None, True)
    gameDisplay.blit(pic.scissors, (display_width*0.6, display_height*0.25))

elif ready_time + 0.5*2 < time < ready_time + 0.5*3:
    init_sound +=1
    text('Paper...', 100, c.white, display_width, display_height // 8, True)
    pygame.draw.circle(gameDisplay, lit, (b_circle1_w, b_circle1_h), 50)
    pygame.draw.circle(gameDisplay, lit, (s_circle1_w, s_circle1_h), 30)
    pygame.draw.circle(gameDisplay, lit, (b_circle2_w, b_circle2_h), 50)
    pygame.draw.circle(gameDisplay, dim, (s_circle2_w, s_circle2_h), 30)
    pygame.draw.circle(gameDisplay, dim, (b_circle3_w, b_circle3_h), 50)
    #pygame.draw.circle(gameDisplay, dim, (s_circle3_w, s_circle3_h), 1)
    text('3', 100, c.black, b_circle1_w, b_circle1_h+5, False, None, True)
    text('2', 100, c.black, b_circle2_w, b_circle2_h + 5, False, None, True)
    text('1', 100, c.black, b_circle3_w, b_circle3_h + 5, False, None, True)
    gameDisplay.blit(pic.paper, (display_width*0.6, display_height*0.25))
    if init_sound == 1:
        #sound.timing()
        sound.countdown2()

elif ready_time + 0.5*3 < time < ready_time + 0.5*4:
    init_sound = 0
    text('Paper...', 100, c.white, display_width, display_height // 8, True)
    pygame.draw.circle(gameDisplay, lit, (b_circle1_w, b_circle1_h), 50)
    pygame.draw.circle(gameDisplay, lit, (s_circle1_w, s_circle1_h), 30)
```

```
        pygame.draw.circle(gameDisplay, lit, (b_circle2_w, b_circle2_h), 50)
        pygame.draw.circle(gameDisplay, lit, (s_circle2_w, s_circle2_h), 30)
        pygame.draw.circle(gameDisplay, dim, (b_circle3_w, b_circle3_h), 50)
        #pygame.draw.circle(gameDisplay, dim, (s_circle3_w, s_circle3_h), 1)
        text('3', 100, c.black, b_circle1_w, b_circle1_h+5, False, None, True)
        text('2', 100, c.black, b_circle2_w, b_circle2_h + 5, False, None, True)
        text('1', 100, c.black, b_circle3_w, b_circle3_h + 5, False, None, True)
        gameDisplay.blit(pic.paper, (display_width*0.6, display_height*0.25))

    elif ready_time + 0.5*4 < time < ready_time + 0.5*5:
        init_sound +=1
        text('Stone!', 100, c.white, display_width, display_height // 8, True)
        pygame.draw.circle(gameDisplay, lit, (b_circle1_w, b_circle1_h), 50)
        pygame.draw.circle(gameDisplay, lit, (s_circle1_w, s_circle1_h), 30)
        pygame.draw.circle(gameDisplay, lit, (b_circle2_w, b_circle2_h), 50)
        pygame.draw.circle(gameDisplay, lit, (s_circle2_w, s_circle2_h), 30)
        pygame.draw.circle(gameDisplay, lit, (b_circle3_w, b_circle3_h), 50)
        #pygame.draw.circle(gameDisplay, dim, (s_circle3_w, s_circle3_h), 1)
        text('3', 100, c.black, b_circle1_w, b_circle1_h+5, False, None, True)
        text('2', 100, c.black, b_circle2_w, b_circle2_h + 5, False, None, True)
        text('1', 100, c.black, b_circle3_w, b_circle3_h + 5, False, None, True)
        gameDisplay.blit(pic.stone, (display_width*0.6, display_height*0.25))
        if init_sound == 1:
            # sound.timing()
            sound.countdown1()

    elif ready_time + 0.5*5 < time < ready_time + 0.5*5 + 0.1:
        init_sound =0
        text('Stone!', 100, c.white, display_width, display_height // 8, True)
        pygame.draw.circle(gameDisplay, lit, (b_circle1_w, b_circle1_h), 50)
        pygame.draw.circle(gameDisplay, lit, (s_circle1_w, s_circle1_h), 30)
        pygame.draw.circle(gameDisplay, lit, (b_circle2_w, b_circle2_h), 50)
        pygame.draw.circle(gameDisplay, lit, (s_circle2_w, s_circle2_h), 30)
        pygame.draw.circle(gameDisplay, lit, (b_circle3_w, b_circle3_h), 50)
        #pygame.draw.circle(gameDisplay, lit, (s_circle3_w, s_circle3_h), 1)
        text('3', 100, c.black, b_circle1_w, b_circle1_h+5, False, None, True)
        text('2', 100, c.black, b_circle2_w, b_circle2_h + 5, False, None, True)
        text('1', 100, c.black, b_circle3_w, b_circle3_h + 5, False, None, True)
        gameDisplay.blit(pic.stone, (display_width*0.6, display_height*0.25))

    elif time > ready_time + 0.5*5 + 0.1:
        text('Stone!', 100, c.black, display_width, display_height // 8, True)
        pygame.draw.circle(gameDisplay, lit, (b_circle1_w, b_circle1_h), 50)
        pygame.draw.circle(gameDisplay, lit, (s_circle1_w, s_circle1_h), 30)
        pygame.draw.circle(gameDisplay, lit, (b_circle2_w, b_circle2_h), 50)
        pygame.draw.circle(gameDisplay, lit, (s_circle2_w, s_circle2_h), 30)
        pygame.draw.circle(gameDisplay, lit, (b_circle3_w, b_circle3_h), 50)
        #pygame.draw.circle(gameDisplay, lit, (s_circle3_w, s_circle3_h), 1)
        text('3', 100, c.black, b_circle1_w, b_circle1_h+5, False, None, True)
        text('2', 100, c.black, b_circle2_w, b_circle2_h + 5, False, None, True)
        text('1', 100, c.black, b_circle3_w, b_circle3_h + 5, False, None, True)
        if init_sound == 1:
            sound.countdownGo()
        detect_now = True
    return detect_now

def pause_menu():
    global display_width, display_height, button_w, button_h, text_size, text_size_1
    while True:
        resizeDisplay()
        gameDisplay.blit(pygame.transform.scale(pic.background, (display_width, display_height)), (0, 0))
        text('Paused', text_size_1, c.white, display_width, display_height//2, True)
        button('center', display_width//4, display_height//3*2, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Main Menu',
text_size, c.black, 'pause', 'intro')
        button('center', display_width//4 * 2, display_height//3*2, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Scan',
text_size, c.black, 'pause', 'scan_screen')
```

```
        button('center', display_width//4 * 3, display_height//3*2, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Play',
text_size, c.black, 'pause', 'game')
        button('center', display_width //3, display_height //6*5, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Instruction',
text_size, c.black, 'pause', 'tutt_1')
        button('center', display_width//3*2, display_height//6*5, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Quit',
text_size, c.black, 'pause', 'quit')
        ut.update_screen(clock)
        ut.quit_x()

def pause_scan_first():
    global display_width, display_height, button_w, button_h, text_size, text_size_1
    sound.scan_first()
    pygame.mixer.music.set_volume(0.3)
    start = time.time()
    while True:
        #resizeDisplay()
        gameDisplay.blit(pygame.transform.scale(pic.background, (display_width, display_height)), (0, 0))
        text('Paused', text_size_1, c.white, display_width, display_height//2, True)
        #text('Please scan your hand first.', text_size, c.white, display_width//2, display_height // 2, False, None, True)
        button('center', display_width//4, display_height//3*2, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Main Menu',
text_size, c.black, 'pause', 'intro')
        button('center', display_width//4 * 2, display_height//3*2, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Scan',
text_size, c.black, 'pause', 'scan_screen')
        button('center', display_width//4 * 3, display_height//3*2, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Play',
text_size, c.black, 'pause', 'scan_screen')
        button('center', display_width //3, display_height //6*5, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Instruction',
text_size, c.black, 'pause', 'tutt_1')
        button('center', display_width//3*2, display_height//6*5, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Quit',
text_size, c.black, 'pause', 'quit')
        ut.update_screen(clock)
        ut.quit_x()
        if time.time()-start > 3:
            pause_menu()

#required to rescale camera image on jetson nano.
def rescale_frame(frame):
    percent = 50
    width = int(frame.shape[1] * percent/100)
    height = int(frame.shape[0] * percent/100)
    dim = (width,height)
    return cv2.resize(frame,dim, interpolation = cv2.INTER_AREA)

#quits everything and exits the game
def quit_all():
    cv2.destroyAllWindows()
    ut.quit_game()
    keras.clear_session()
    pygame.quit()

def frame_adjust(frame, isit_scan=False):
    global display_width, display_height, button_w, button_h, text_size, text_size_1
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    if isit_scan:
        frame = translucent_obj(frame, draw_rect_right, 1.0)
    else:
        #cv2.rectangle(frame, (50, 300), (175, 150), (0, 255, 0), 5)
        cv2.rectangle(frame, (10, 355), (213, 125), (0, 255, 0), 5)
    frame = np.rot90(frame)
    x_pos = (display_width // 2 - display_width // 24 - frame.shape[0])//2
    y_pos = display_height//4 + (display_height - display_height//4 - display_height//9*2 - frame.shape[1])//2
    frame = pygame.surfarray.make_surface(frame)
    gameDisplay.blit(frame, (x_pos, y_pos))

def hand_scan_screen():
    global display_width, display_height, button_w, button_h, text_size, text_size_1
```

```
    global hand_hist
    global cap
    complete = 0
    global scanned
    hand_hist = None
    scanned = 0
    #resizeDisplay()
    while complete == 0:
        #resizeDisplay()
        ret, frame = cap.read()
        frame = rescale_frame(frame)
        init_game_screen()
        frame_adjust(frame, True)
        text('Place your hand on the boxes and press "Scan".', 50, c.white, display_width, display_height//4, True)
        #hand_x = display_width // 4 + display_width // 24 + (display_width - (display_width // 2 + display_width // 24 + 288))//2
        gameDisplay.blit(pic.hand, (display_width//4 * 3, display_height//4))
        pressed = button('center', display_width//2, display_height-100, 300, 100, c.red_dim, c.red_lit, 'Scan', 50, c.white,
'scan_screen', 'hand_scan')
        if pressed == 1:
            hand_hist = hand_scan(frame)
            scanned = 1
            game_screen()
            complete = 1
        ut.update_screen(clock)
        ut.quit_x()

def draw_rect_right(frame, return_list_only=False, color=(200, 0, 255), thickness=2):
    '''Draw rectangles'''
    y, x, channel = frame.shape
    num_box = 12
    start_rect = []
    start = [x * 0.10, y * 0.45] #0.10, 0.35
    end = [x * 0.20, y * 0.65] #0.23, 0.65
    spacing = [((end[0] - start[0]) - (3 * 10)) // 2, ((end[1] - start[1]) - (4 * 10)) // 3]
    count_x = 0
    count_y = 0

    for i in range(1, num_box + 1):
        start_rect.append([int(start[0] + count_x * (10 + spacing[0])), int(start[1] + count_y * (10 + spacing[1]))])
        count_x = count_x + 1
        if i % 3 == 0:
            count_x = 0
            count_y = count_y + 1
    start_rect = np.array(start_rect)
    end_rect = start_rect + 10

    #cv2.imwrite("image.jpg", frame)

    if not return_list_only:
        for i in range(0, num_box):
            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
    else:
        return start_rect, end_rect

def translucent_obj(frame, action=None, alpha=0.5):
    '''Put a translucent object in the frame'''
    img = frame.copy()
    action(img)
    frame = cv2.addWeighted(img, alpha, frame, 1 - alpha, 0)
    return frame

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)
```

```
    roi = np.zeros([120, 10, 3], dtype=hsv.dtype)
    size = int(start_rect.size / 2)
    for i in range(size):
        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]]
        # (H,S,V)
        # max(180, 255, 255)
    #         Just for checking
    #         if i == 0:
    #             print(roi)
    #             print('end')
    #             print(hsv[start_rect[i][1]:end_rect[i][1], start_rect[i][0]:end_rect[i][0]])

    hand_hist = cv2.calcHist([roi], [0, 1], None, [180, 256], [0, 180, 0, 256])
    cv2.normalize(hand_hist, hand_hist, 0, 255, cv2.NORM_MINMAX)
    return hand_hist

def hand_scan(frame):
    start_rect, end_rect = draw_rect_right(frame, True)
    hand_hist = set_hand_hist(frame, start_rect, end_rect)
    return hand_hist

def crop_binary(frame):
    crop = frame[125:355, 10:213]  # y, x
    thresh = hist_mask(crop, hand_hist)
    clear = clearer(thresh, 5, 1)
    return clear

def hist_mask(frame, hand_hist):
    '''Return binary image of the skin'''
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    dst = cv2.calcBackProject([hsv], [0, 1], hand_hist, [0, 181, 0, 256], 1)
    disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
    cv2.filter2D(dst, -1, disc, dst)
    ret, thresh1 = cv2.threshold(dst, 0, 255, 0)
    return thresh1

def clearer(frame, kernel_size=5, iteration=1):
    '''cover the holes in the image'''
    kernel = np.ones((kernel_size,kernel_size),np.uint8)
    frame = cv2.erode(frame,kernel,iterations = iteration)
    frame = cv2.morphologyEx(frame, cv2.MORPH_OPEN, kernel)
    frame = cv2.morphologyEx(frame, cv2.MORPH_CLOSE, kernel)
    return frame

def nonzero_ratio(clear):
    h_clear = cv2.resize(clear, (100, 100))
    nonzero = cv2.countNonZero(h_clear)
    nonzero_ratio = (nonzero / (100 * 100)) * 100
    return nonzero_ratio, h_clear

def predict(h_clear):
    np_clear = h_clear.reshape(-1, 100, 100, 1)
    np_clear = np.array(np_clear)
    np_clear = np_clear / 255
    prediction = model.predict([np_clear])
    predictions = np.where(prediction == np.amax(prediction))
    # gesture = ''
    if predictions[1] == 0 and (prediction[0][0] >= 0.9):
        gesture = 'Stone'
    elif predictions[1] == 1 and (prediction[0][1] >= 0.9):
        gesture = 'Paper'
    elif predictions[1] == 2 and (prediction[0][2] >= 0.9):
        gesture = 'Scissors'
    else:
        gesture = '???'
```

```
    return gesture

def detect(frame, detect_now):
    if not detect_now:
        pass
    else:
        clear = crop_binary(frame)
        white_ratio, h_clear = nonzero_ratio(clear)
        if white_ratio > 14:
            gesture = predict(h_clear)
        else:
            gesture = 'Nothing'
        return gesture


def display_gesture(gesture, rand_gesture, frame):
    global invalid_gesture
    global display_width, display_height, button_w, button_h, text_size, text_size_1
    start = time.time()
    result_me_int = int(0)
    result_com_int = int(0)
    result_me, result_com = win_lose(gesture, rand_gesture)
    if result_me == 'You Win!':
        sound.win()
    elif result_me == 'Draw':
        sound.draw()
    elif result_me == 'You Lose':
        sound.lose()
    while True:
        #resizeDisplay()
        #frame = rescale_frame(frame)
        init_game_screen()
        frame_adjust(frame)
        text(gesture, text_size_1, c.white, display_width*0.15, display_height*0.8)
        text(rand_gesture, text_size_1, c.white, display_width*0.75, display_height*0.8)
        if rand_gesture == 'Scissors':
            gameDisplay.blit(pic.scissors, (display_width*0.6, display_height*0.25))
        elif rand_gesture == 'Paper':
            gameDisplay.blit(pic.paper, (display_width*0.6, display_height*0.25))
        elif rand_gesture == 'Stone':
            gameDisplay.blit(pic.stone, (display_width*0.6, display_height*0.25))
        text(result_me, int(text_size_1*1.5)  , c.white, display_width*0.5, display_height*0.13, False, None, True)
        text('You: ', int(text_size_1*0.5), c.red_dim, display_width*0.03, display_height*0.8, False, c.white)
        text('Com: ', int(text_size_1*0.5), c.blue_dim, display_width*0.6, display_height*0.8, False, c.white)

        if result_me == 'You Win!':
            result_me_int = int(1)
            invalid_gesture = int(0)

        elif gesture == "???":
            result_me_int = int(0)
            result_com_int = int(0)
            invalid_gesture = int(1)

        elif gesture == "Nothing":
            result_me_int = int(0)
            result_com_int = int(0)
            invalid_gesture = int(1)

        elif result_me == 'Draw':
            result_me_int = int(0)
            invalid_gesture = int(1)

        elif result_me == 'You Lose':
            result_me_int = int(0)
            invalid_gesture = int(0)
```

```
        if result_com == 'Win':
            result_com_int = int(1)
            invalid_gesture = int(0)
        elif result_com == 'Draw':
            result_com_int = int(0)
            invalid_gesture = int(1)

        elif result_com == 'Lose':
            result_com_int = int(0)
            invalid_gesture = int(0)

        button('top right', display_width, 0, 50, 50, c.light_blue_dim, c.light_blue_lit, '', '', '', 'game', 'paused')
        pygame.draw.rect(gameDisplay, c.black, (display_width - 40, 10, 10, 30))
        pygame.draw.rect(gameDisplay, c.black, (display_width - 20, 10, 10, 30))

        ut.update_screen(clock)
        ut.quit_x()
        time.sleep(2)
        break
    return result_me_int, result_com_int


def win_lose(gesture, rand_gesture):
    result_me = ''
    result_com = ''
    if gesture == rand_gesture:
        result_me = result_com = 'Draw'
    elif gesture == 'Scissors' and rand_gesture == 'Paper':
        result_me = 'You Win!'
        result_com = 'Lose'

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

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

    elif gesture == 'Stone' and rand_gesture == 'Scissors':
        result_me = 'You Win!'
        result_com = 'Lose'

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

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

    elif gesture == 'Nothing':
        result_me = 'No Result'
        result_com = 'No Result'

    elif gesture == '???':
        result_me = 'No Result'
        result_com = 'No Result'

    return result_me, result_com

def init_game_screen():
    global display_width, display_height, button_w, button_h, text_size, text_size_1

    gameDisplay.blit(pygame.transform.scale(pic.background2, (display_width, display_height)), (0, 0))
    #button('top right', display_width, 0, 50, 50, c.light_blue_dim, c.light_blue_lit, '||', 30, c.black, 'game', 'paused')
```

```
    button('top right', display_width, 0, 50, 50, c.light_blue_dim, c.light_blue_lit, '', '', '', 'game', 'paused')
    pygame.draw.rect(gameDisplay, c.black, (display_width - 40, 10, 10, 30))
    pygame.draw.rect(gameDisplay, c.black, (display_width - 20, 10, 10, 30))

def random_gesture():
    gesture_int = random.randint(0, 2)
    if gesture_int == 0:
        gest_ture = 'Scissors'
    if gesture_int == 1:
        gest_ture = 'Paper'
    if gesture_int == 2:
        gest_ture = 'Stone'
    return gest_ture


def intermediate_gesture(gest_ture):
    gesture = ["Stone", "Paper", "Scissors"]
    continuePlaying = True
    prevGesture = ""
    reroll = ""
    #result = ""
    #alt = 0
    #numoff = 0
    choice = 3
    start = time.time()
    #score_me = int(0)
    #score_com = int(0)
    #while True:
    if gest_ture == 'Stone':
        gesture_integer = 0
    elif gest_ture == 'Paper':
        gesture_integer = 1
    elif gest_ture == 'Scissors':
        gesture_integer = 2
    elif gest_ture == 'Nothing':
        gesture_integer = 2
    elif gest_ture == '???' :
       gesture_integer = 2


    machine_int = random.randint(0, 2)
    if machine_int == 0:
        machine_gesture = 'Stone'
    elif machine_int == 1:
        machine_gesture = 'Paper'
    elif machine_int == 2:
        machine_gesture = 'Scissors'

    if gest_ture == machine_gesture:
        result = 'Draw'
    elif gest_ture == 'Scissors' and machine_gesture == 'Paper':
        result = 'You Win!'
    elif gest_ture == 'Stone' and machine_gesture == 'Paper':
        result= 'You Lose'
    elif gest_ture == 'Paper' and machine_gesture == 'Scissors':
        result = 'You Lose'
    elif gest_ture == 'Stone' and machine_gesture == 'Scissors':
        result ='You Win!'
    elif gest_ture == 'Paper' and machine_gesture == 'Stone':
        result = 'You Win!'
    elif gest_ture == 'Scissors' and machine_gesture == 'Stone':
        result = 'You Lose'
    elif gest_ture == 'Nothing':
        result = 'You Lose'
    elif gest_ture == '???':
        result = 'You Lose'
```

```
    if result == "You Win!":
        reroll = random.randint(0, 1)
        if reroll == 0:
            machine_int = random.randint(0, 2)
        elif reroll == 1:
            machine_int == machine_int
    elif (result == "Draw"):
        if reroll == 0:
            machine_int = random.randint(0, 2)
        elif reroll == 1:
            machine_int == machine_int
    elif (result == "You Lose"):
        machine_int == machine_int

    if machine_int == 0:
        machine_gesture = 'Stone'
    elif machine_int == 1:
        machine_gesture = 'Paper'
    elif machine_int == 2:
        machine_gesture = 'Scissors'

    '''#print(machine_integer)
    prevGesture = gesture_integer
    print(prevGesture)
    return streak
    machine_int = prevGesture - 2
    if (machine_int < 0):
        machine_int += 3
        if (result == "You Win!"):
            streak += 1
        elif (result == "You Lose"):
            streak == streak
        elif (result == "Draw"):
            streak == streak'''

    return machine_gesture


def hard_gesture(gest_ture):
    global probabilitiesRPS
    choices = ["Rock","Paper","Scissors"]
    choi = ['r','p','s']
    continuePlaying = True
    prevGesture = ""
    result = ""
    choice = 3
    probRock = 0
    probPaper = 0
    probScissors = 0

    machineChoice = random.randint(0, 2)

    if gest_ture == machineChoice:
        result = 'Draw'
    elif gest_ture == 'Scissors' and machineChoice == 'Paper':
        result = 'You Win!'
    elif gest_ture == 'Stone' and machineChoice == 'Paper':
        result= 'You Lose'
    elif gest_ture == 'Paper' and machineChoice == 'Scissors':
        result = 'You Lose'
    elif gest_ture == 'Stone' and machineChoice == 'Scissors':
        result ='You Win!'
    elif gest_ture == 'Paper' and machineChoice == 'Stone':
        result = 'You Win!'
    elif gest_ture == 'Scissors' and machineChoice == 'Stone':
```

```
        result = 'You Lose'
    elif gest_ture == 'Nothing':
        result = 'You Lose'
    elif gest_ture == '???':
        result = 'You Lose'

    if gest_ture == 'Stone':
        gesture_integer = 0
    elif gest_ture == 'Paper':
        gesture_integer = 1
    elif gest_ture == 'Scissors':
        gesture_integer = 2
    elif gest_ture == 'Nothing':
        gesture_integer = 2
    elif gest_ture == '???' :
        gesture_integer = 2

    prevGesture = gesture_integer

    transMatrix = buildTransitionProbabilities(prevGesture,gesture_integer,result)
    machineChoice = random.randint(1, 100)
    probabilitiesRPS[0] = transMatrix[prevGesture][0]
    probabilitiesRPS[1] = transMatrix[prevGesture][1]
    probabilitiesRPS[2] = transMatrix[prevGesture][2]
    rangeR = probabilitiesRPS[0] * 100
    rangeP = probabilitiesRPS[1] * 100 + rangeR
    if (machineChoice <= rangeR):
        machineChoice = 1
    elif (machineChoice <= rangeP):
        machineChoice = 2
    else:
        machineChoice = 0

    if machineChoice == 0:
        machinechoice = 'Stone'
    elif machineChoice == 1:
        machinechoice = 'Paper'
    elif machineChoice == 2:
        machinechoice = 'Scissors'

    print("Your winning transition matrix is:\nr: %s\np: %s\ns: %s\n" % (tMatrix[0],tMatrix[1],tMatrix[2]))
    print("Your losing transition matrix is:\nr: %s\np: %s\ns: %s\n" % (tMatrixL[0],tMatrixL[1],tMatrixL[2]))
    print("Your tying transition matrix is:\nr: %s\np: %s\ns: %s\n" % (tMatrixT[0],tMatrixT[1],tMatrixT[2]))
    return machinechoice



def buildTransitionProbabilities(pC,c,winloss):
    global buildTMatrix
    global buildTMatrixL
    global buildTMatrixT
    choi = ['r','p','s']

    if winloss == "Win!":
        for i, x in buildTMatrix.items():
            if ('%s%s' % (choi[pC],choi[c]) == i):
                buildTMatrix['%s%s' % (choi[pC], choi[c])] += 1
    elif winloss == "Draw":
        for i, x in buildTMatrixT.items():
            if ('%s%s' % (choi[pC],choi[c]) == i):
                buildTMatrixT['%s%s' % (choi[pC], choi[c])] += 1
    else:
        for i, x in buildTMatrixL.items():
            if ('%s%s' % (choi[pC],choi[c]) == i):
                buildTMatrixL['%s%s' % (choi[pC], choi[c])] += 1
```

```
    return buildTransitionMatrix(winloss)

def buildTransitionMatrix(winlosstwo):
    global tMatrix
    global tMatrixL
    global tMatrixT

    if winlosstwo == "Win!":
        rock = buildTMatrix['rr'] + buildTMatrix['rs'] +buildTMatrix['rp']              #number of gesture that appeared
        paper = buildTMatrix['pr'] + buildTMatrix['ps'] +buildTMatrix['pp']
        scissors = buildTMatrix['sr'] + buildTMatrix['ss'] +buildTMatrix['sp']
        choi = ['r','p','s']
        for row_index, row in enumerate(tMatrix):
            for col_index, item in enumerate(row):
                a = int(buildTMatrix['%s%s' % (choi[row_index],choi[col_index])])
                if (row_index == 0):
                    c = a/rock
                elif (row_index == 1):
                    c = a/paper
                else:
                    c = a/scissors
                row[col_index] = float(c)
        return (tMatrix)

    elif winlosstwo == "Draw":
        rock = buildTMatrixT['rr'] + buildTMatrixT['rs'] +buildTMatrixT['rp']
        paper = buildTMatrixT['pr'] + buildTMatrixT['ps'] +buildTMatrixT['pp']
        scissors = buildTMatrixT['sr'] + buildTMatrixT['ss'] +buildTMatrixT['sp']
        choi = ['r','p','s']
        for row_index, row in enumerate(tMatrixT):
            for col_index, item in enumerate(row):
                a = int(buildTMatrixT['%s%s' % (choi[row_index],choi[col_index])])
                if (row_index == 0):
                    c = a/rock
                elif (row_index == 1):
                    c = a/paper
                else:
                    c = a/scissors
                row[col_index] = float(c)
        return (tMatrixT)

    else:
        rock = buildTMatrixL['rr'] + buildTMatrixL['rs'] +buildTMatrixL['rp']
        paper = buildTMatrixL['pr'] + buildTMatrixL['ps'] +buildTMatrixL['pp']
        scissors = buildTMatrixL['sr'] + buildTMatrixL['ss'] +buildTMatrixL['sp']
        choi = ['r','p','s']
        for row_index, row in enumerate(tMatrixL):
            for col_index, item in enumerate(row):
                a = int(buildTMatrixL['%s%s' % (choi[row_index],choi[col_index])])
                if (row_index == 0):
                    c = a/rock
                elif (row_index == 1):
                    c = a/paper
                else:
                    c = a/scissors
                row[col_index] = float(c)
        return (tMatrixL)

def expert_gesture(gest_ture):

    if gest_ture == 'Stone':
        machinechoice = 'Paper'
    elif gest_ture == 'Paper':
        machinechoice = 'Scissors'
```

```
    elif gest_ture == 'Scissors':
        machinechoice = 'Stone'
    elif gest_ture == 'Nothing':
        machinechoice = random.randint(0, 2)
        if machinechoice == 0:
            machinechoice = 'Scissors'
        if machinechoice == 1:
            machinechoice = 'Paper'
        if machinechoice == 2:
            machinechoice = 'Stone'
    elif gest_ture == '???' :
        machinechoice = random.randint(0, 2)
        if machinechoice == 0:
            machinechoice = 'Scissors'
        if machinechoice == 1:
            machinechoice = 'Paper'
        if machinechoice == 2:
            machinechoice = 'Stone'

    return machinechoice

def game_screen():
    music.stop()
    music.game()
    global cap
    global detect_now
    global matches
    matches = 0
    global action
    global rand_gesture
    global invalid_gesture
    global display_width, display_height, button_w, button_h, text_size, text_size_1
    global gameDifficulty
    complete = 0
    start = time.time()
    score_me = int(0)
    score_com = int(0)
    while complete == 0:
        # frame to display
        #resizeDisplay()
        ret, frame = cap.read()
        mouse = pygame.mouse.get_pos()
        frame = rescale_frame(frame)
        init_game_screen()
        frame_adjust(frame)
        passed = time.time()
        detect_now = SPS_timing(passed-start)
        gesture = detect(frame, detect_now)

        if detect_now:
            if gameDifficulty == 'easy':
                rand_gesture = random_gesture()
            elif gameDifficulty == 'intermediate':
                rand_gesture = intermediate_gesture(gesture)
            elif gameDifficulty == 'hard':
                rand_gesture = hard_gesture(gesture)
            elif gameDifficulty == 'expert':
                rand_gesture = expert_gesture(gesture)

            win_lose_me, win_lose_com = display_gesture(gesture, rand_gesture, frame)
            score_me = int(score_me) + int(win_lose_me)
            score_com = int(score_com) + int(win_lose_com)
            if(invalid_gesture == 0):
                matches += 1
            start = time.time()
```

```
            if gesture == 'Stone':
                cv2.imwrite('/home/nyp/Desktop/pictures/Stone/stone_{0}.jpg'.format(timestr), clear)

            elif gesture == 'Paper':
                cv2.imwrite('/home/nyp/Desktop/pictures/Paper/paper_{0}.jpg'.format(timestr), clear)

            elif gesture == 'Scissors':
                cv2.imwrite('/home/nyp/Desktop/pictures/Scissors/scissors_{0}.jpg'.format(timestr), clear)

            detect_now = False
        if matches == 5:
            complete = 1
            matches = 0
            result_screen(score_me, score_com)

        ut.update_screen(clock)
        clear = crop_binary(frame)
        timestr = time.strftime("%H%M%S")
        ut.quit_x()


def result_screen(score_me, score_com):
    global display_width, display_height, button_w, button_h, text_size, text_size_1

    if score_me == score_com:
        music.stop()
        music.draw()
    elif score_me > score_com:
        music.stop()
        music.win()
    elif score_com > score_me:
        music.stop()
        music.lose()

    while True:
        gameDisplay.blit(pygame.transform.scale(pic.background, (display_width, display_height)), (0, 0))

        if score_me == score_com:
            text('Draw? No fun here.', text_size_1, c.white, display_width//2, display_height//10, False, None, True)
            text('Your Score:', text_size, c.white, display_width//3, 300, False, None, True)
            text(str(score_me), text_size, c.white, display_width//3, 400, False, None, True)
            text('Computer Score:', text_size, c.white, display_width // 3*2, 300, False, None, True)
            text(str(score_com), text_size, c.white, display_width // 3*2, 400, False, None, True)
        elif score_me > score_com:
            text('Tsk. You Win.', text_size_1, c.white, display_width//2, display_height//10, False, None, True)
            text('Your Score:', text_size, c.white, display_width//3, 300, False, None, True)
            text(str(score_me), text_size, c.white, display_width//3, 400, False, None, True)
            text('Computer Score:', text_size, c.white, display_width // 3*2, 300, False, None, True)
            text(str(score_com), text_size, c.white, display_width // 3*2, 400, False, None, True)
        elif score_com > score_me:
            text('Ha-Hah! You Lose!', text_size_1, c.white, display_width//2, display_height//10, False, None, True)
            text('Your Score:', text_size, c.white, display_width//3, 300, False, None, True)
            text(str(score_me), text_size, c.white, display_width//3, 400, False, None, True)
            text('Computer Score:', text_size, c.white, display_width // 3*2, 300, False, None, True)
            text(str(score_com), text_size, c.white, display_width // 3*2, 400, False, None, True)\

        button('center', display_width//4, display_height//4*3, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Main Menu',
text_size, c.black, 'result', 'intro')
        button('center', display_width//4 * 2, display_height//4*3, button_w + 15, button_h, c.light_blue_dim, c.light_blue_lit,
'Choose Mode', text_size, c.black, 'result', 'play')
        button('center', display_width//4 * 3, display_height//4*3, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Play again',
text_size, c.black, 'result', 'game_1')
        button('center', display_width//4 * 2, display_height//7*6, button_w, button_h, c.light_blue_dim, c.light_blue_lit, 'Quit',
text_size, c.black, 'result', 'quit')
        text('If the lighting has changed or the next player is a different person,', int(text_size*0.75), c.black, display_width//2,
```

```
display_height-60, False, c.light_blue_dim, True)
        text("please select 'Scan again' to play again.", int(text_size*0.75), c.black, display_width//2, display_height-30, False,
c.light_blue_dim, True)
        ut.update_screen(clock)
        ut.quit_x()

check_camera()
```