TECHNICAL MANUAL FOR THE REINHERIT IMMERSIVE PERFORMANCE

SUMMARY

This document presents a short description for the setting up and usage of the source code developed to support the first immersive performance for the Reinherit project. The purpose of the document is to facilitate users who wish to recreate a similar experience to their own premises.

The code for the application can be accessed from the following GitHub repository link: https://github.com/CYENS/Reinherit-Hadjigeorgakis-Kornesios-Mansion

Contents

1 Guidelines for reuse

These are the proposed steps to be followed when setting up a system for such a performance:

a) The mobile phones that contain the Station application should be placed at elevated positions where they can maintain a good FOV. Also, visitors should not be able to reach them and move them out of position since this will affect the functionality of the application. Also if it is possible, the devices should include locking mechanisms to avoid theft.

b) The Bluetooth speakers should also be hidden from the eyes of the visitors and placed near points of interest. This will make the exhibition more interactive and engaging.

c) When every mobile phone and Bluetooth speaker is set up, the Remote application must be used to configure each Station. Primarily, set the "base" image, configure the volume, and define the weights to be used for the detection.

d) If during the performance something inside the room is changed (lights, objects moved, etc.), a reset must be performed to the Station application.

It is important to mention that data collected during the event do not contain any sensitive private information about the visitors or the performers. The application only tracks the number of people passing in front of the cameras and the number of the performers that are located inside a room at any given time. Also, the stations do not record any video nor have the ability to do so which protects the anonymity of all guests during the event.

More details regarding the specific app will be provided in dedicated sections of the Reinherit digital hub with manuals, instructions, guidelines and links to the source code for the recreation of the experience, in a more "personalized" context for other potential cultural heritage sites across Europe.

2 Technical Guidelines

The code for the application can be accessed from the following GitHub repository link: https://github.com/CYENS/Reinherit-Hadjigeorgakis-Kornesios-Mansion

In this section we are going to describe some important features of both the applications and review parts of their code and the necessary files that are needed in order to re-create a similar experience.

2.1.1 Setting up the Base Image

Firstly, the Base Image need to be setup. It is the base image that the Base Station will compare every frame when the exhibition starts. By detecting differences in pixel values, it would be able to estimate how many people are in front of the camera and will play the appropriate sounds. The Base Image is taken before the exhibition so that no visitors are present in the picture. Another important step that needs to be taken in order to correctly setup the Base Station is to calibrate three more parameters:

1. The Weight.
2. The Minimum Threshold.
3. The Maximum Value.

The weight describes how sensitive the Base Station would be in detecting rapid changes in the pixel values. Its value ranges from 0 to 1 and by default is set to 0.5. By changing this value, the base station gets more sensitive in detecting people moving in front of the camera or it gives more weight in comparing the Base image with the current frame. The Minimum Threshold determines the least amount of pixel values that need to be changed to register a visitor while the Maximum Value is the upper limit of pixels.

2.1.2 The "Pixel comparison algorithm"

The Base Station application uses a pixel comparison algorithm that determines the number of visitors located in a room. The implementation of the algorithm can be found in the "CameraInputManager" class. The algorithm has the following steps:

1. Setup the Base Image. It is required in order for the algorithm to be able to do comparisons.
2. Get a new frame from the camera.
3. Convert the frame to grayscale and rescale it.
4. Check the value of every pixel in the new frame with the value of every pixel in the Base Image and the previous frame that was received from the camera.
5. Determine how many visitors are in the room.
6. Repeat for the next frame.

The grayscale conversion and rescale of the processed frame is done to make the pixel comparison between the two images faster since we are checking every frame the Base station's camera provides. Based on the

previous parameters (Weight, Minimum Threshold and Maximum Value), the algorithm then determines the number of visitors and plays certain number of sounds. The Base Stations need to be loaded with the sound files beforehand, and they all follow a specific name format, “sound” plus a number between one to five.

Figure 1 Sound files in the Base Station using the specific name format with the prefix “sound”.

The sound file with the number zero is the default background sound that is always playing and acts as ambiance. The rest of the sounds are played simultaneously when there are visitors present. The number in the sound file name represents the number of visitors that should be in the room for the sound to be played.

2.1.3 The “SoundManager”class

To play all these sounds at the same time we use the “SoundManager” class which creates a new thread each time a sound needs to be played.

```java
//MusicThread class that extends java Thread. Run multiple instances of this class to play simultaneously multiple sources of audio
public static class MusicThread extends Thread{
    public MediaPlayer mSoundPlayer;
    private boolean mIsRunning;
    private final boolean mLoopSound;

    public MusicThread(int id, String filePrefix, String fileExtension, Activity parentActivity, boolean loop){
        this.mSoundPlayer = createMediaPlayer(id, filePrefix, fileExtension, parentActivity);
        mLoopSound = loop;
    }

    public void stopPlayback() { this.mIsRunning = false; }

    //Main function that runs everytime a new thread is created
    public void run(){
        if (mSoundPlayer == null) return;

        this.mIsRunning = true;

        mSoundPlayer.start();
        mSoundPlayer.setLooping(mLoopSound);

        try {
            while(mSoundPlayer.isPlaying() && mIsRunning){
                Thread.sleep( millis: 100); // wait
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        mSoundPlayer.release();
    }
}
```

Figure 2 The "MusicThread" class creates a new thread each time a new sound needs to be played. The sound will stop playing when the "stopPlayback" function is called.

2.1.4 The "Remote station"

Both applications, Base Station and Remote, communicate by using the Bluetooth protocol which enables them to exchange messages between them. Primarily, the Remote sends most of the requests to the base station to calibrate it since most of the time the stations are in difficult to reach positions. In order to create new requests to be sent from the Remote, one can edit the "PerformanceController" class and add new message requests by adding listeners to new or existing buttons. Similarly, by editing the "MainActivity" class, we can write new functions or edit the existing ones that will execute every time we receive a specific message in the Base Station.

```java
Button btn1 = (Button) view.findViewById(R.id.startButton);
btn1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        try {
            Toast.makeText(getActivity(), text: "Starting...", Toast.LENGTH_LONG).show();
            mViewModel.sendMessage(new JSONObject().put( name: "id", Constants.COMMANDS.START).put( name: "val", value: "0").toString());
        } catch (JSONException e) {
            e.printStackTrace();
        }

    }
});
```

Figure 3 Adding a button that will send a message to the Base Station to start monitoring a room.

```
switch (id) {
    case GET_STATUS:
        //mService.write(mStatus.getText().toString().getBytes()); <--new code follows. Uncomment this if needed
        mTrackerText = findViewById(R.id.textViewTracker);
        String textToSend = "s"+mTrackerText.getText();
        mBluetoothManager.write(textToSend.getBytes());

        //save a screenshot of the camera
        ImageUtilities.saveBitmap(mPreviewView.getBitmap(), label: "gs_");
        break;
    case RESET_CAMERA_POSE:
        break;
    case SET_FREQ:
        //set the frequency of the test sound and adjust the position of the slider UI
        Slider slider = findViewById(R.id.sliderFreq);
        slider.setValue(valueInt);
        break;
    case START:
        //start all sounds
        soundStopped = false;
        break;
```

Figure 4 Handling the messages that can be received from the Remote.

Besides the calibration, the Remote can control the volume of the Base Stations. It can also start or stop the camera detection and it can request from the Base Station to send the number of visitors that are currently in front of the camera along with all the current parameters.

Another aspect that the Remote is controlling is the number of musicians that are currently present in a room. Each Base Station is typically associated with one room and with the Remote someone can register manually the number of musicians that are in it and save that information in .csv logs which are created at the Base Station. Along with the number of musicians, the Base Station automatically saves the Day of the recordings, the station's parameters and the number of visitors that pass through.

2.1.5 Setup of the specific performance

For the Hadjigeorgakis Kornesios Mansion event, there were 14 rooms in total which were in two floors. The Remote application can be edited to accommodate more than two floors and more rooms for each occasion.