手势识别项目总结

项目结构：

分为 Android 客户端和 WPF windows 服务端。

版本控制：

GitHub

Repository 地址：

Android 端：https://github.com/BigDipper7/Gesture-Recognition-Android-Client-Side
WPF 端：https://github.com/BigDipper7/Gesture-Recognition-WPF-Server-Side

WPF 服务端：

功能：

1. 根据 Android 端传来数据显示对应动画并播放音效
2. 根据传感器手势显示对应动画并播放音效

采用技术：

1. 采用 WPF 来制作全屏显示的 windows 应用程序。
2. 动画采用 Storyboard 和 DoubleAnimation 来实现动画显示，DoubleAnimation 控制黑色幕布进行伸缩遮盖天空背景。
3. 服务端客户端通讯采用以 WPF 作为服务端的方式的 WebSocket 进行消息通讯，消息体为纯文本，未做加密。
4. 声音播放采用线程安全的 MediaPlayer 的方式，可播放多种格式。
5. 状态转化控制选用状态机

关键技术介绍：

1. 采用 WPF 进行 windows 编程，根据需求，在程序启动时设置为：程序全屏、永远保持在其他应用最前端。同时整体程序画布设置为固定的 1280 * 800 的大小，为 landscape 的方式。

```
public MainWindow()
{
    InitializeComponent();

    this.Topmost = true;
    this.Hide();
    this.Show();

    refreshStoryBoard();
    Thread eneterThread = new Thread(new Thread
    eneterThread.Start();
```

2. 采用 Storyboard 和 DoubleAnimation，制作幕布遮盖和开启的线性动画。其中为了处理方便以及考虑到 Storyboard 动画控制端的特殊性，同时设置了开启和关闭两个动画，经由状态机根据命令来处理动画的 start、stop、pause、resume 的操作，同时需要注意的是当动画 resume 时会根据开启动画方向不同需要同时设定动画的 starttime。

```
<Storyboard x:Key="sbopening">
    <DoubleAnimation Storyboard.TargetName="rectMask"
  Storyboard.TargetProperty="Height" From="800" To="0"
  Duration="0:0:10" BeginTime="0:0:0"/>
</Storyboard>
<Storyboard x:Key="sbclosing">
    <DoubleAnimation Storyboard.TargetName="rectMask"
  Storyboard.TargetProperty="Height" From="0" To="800"
  Duration="0:0:10" BeginTime="0:0:0"/>
</Storyboard>
</Window.Resources>
```

3. Android 端和 WPF 端采用 WebSocket 的通讯方式。消息体数据采用纯文本，不加密。建立以 WPF 作为服务端的中心网络，支持一机多联，同时由于状态机存在保证多用户之间的操作的安全性。

   其中 WPF 端 websocket 采用了 Fleck 框架，简单快捷，在启动 windows 窗体的时候同时开子线程启动 Websocket 服务器，但是也需要用特殊手段在子线程中通知前端 windows 窗体的动画改变。

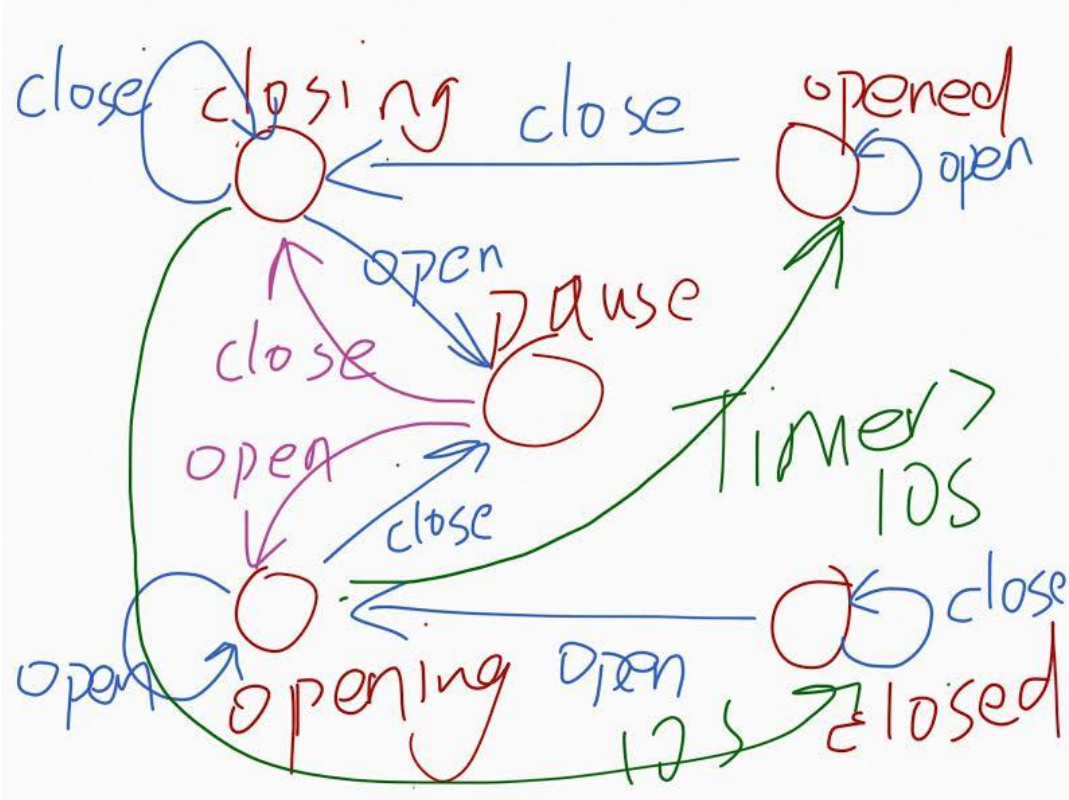
```
public MainWindow()
{
    InitializeComponent();

    this.Topmost = true;
    this.Hide();
    this.Show();

    refreshStoryBoard();
    Thread eneterThread = new Thread(new ThreadStart(runWebSocketServer));
    eneterThread.Start();

    Thread portThread = new Thread(new ThreadStart(initSerialPortInfo));
    portThread.Start();

    mediaPlayer.Open(new Uri("d:/test.mp3", UriKind.Absolute));
    mediaPlayer.MediaEnded += new EventHandler(Media_Ended);
    //mediaPlayer.Play();
}
```

```
public void runWebSocketServer()
{
    var server = new WebSocketServer("ws://0.0.0.0:8090");
    server.RestartAfterListenError = true;
    server.Start(socket =>
    {
        socket.OnOpen = () =>
            {
                Console.WriteLine("Open!");
            };
        socket.OnClose = () =>
            {
                Console.WriteLine("Close!");
            };
        socket.OnMessage = message =>
            {
                socket.Send("received : " + message);
                Console.WriteLine("receive msg..." + message);

                String msg = message;
                if (msg.Equals("Open"))
                {
                    Console.WriteLine("Simulate Open Action");
                    simulateOpenAction();
                }
                else if (msg.Equals("Close"))
                {
                    Console.WriteLine("Simulate Close Action");
                    simulateCloseAction();
                }
            };
    });

    Console.WriteLine("Press any key to continue...");
    Console.WriteLine();
    Console.ReadLine();

    mySerialPort.Close();
}
```

曾采用 Eneter 框架实现.Net 和 Android 端通讯，但是由于框架的极其不稳定，重构为 Fleck 框架。

```
private IDuplexTypedMessageReceiver<MyResponse, MyRequest> myReceiver;

public void run()
{
    // Create message receiver receiving 'MyRequest' and receiving 'MyResponse'.
    IDuplexTypedMessagesFactory aReceiverFactory = new DuplexTypedMessagesFactory();
    myReceiver = aReceiverFactory.CreateDuplexTypedMessageReceiver<MyResponse, MyRequest>();

    // Subscribe to handle messages.
    myReceiver.MessageReceived += OnMessageReceived;

    // Create TCP messaging.
    IMessagingSystemFactory aMessaging = new TcpMessagingSystemFactory();
    IDuplexInputChannel anInputChannel
        = aMessaging.CreateDuplexInputChannel("tcp://192.168.43.167:8067/");
    //= aMessaging.CreateDuplexInputChannel("tcp://192.168.173.1:8060/");

    // Attach the input channel and start to listen to messages.
    myReceiver.AttachDuplexInputChannel(anInputChannel);

    Console.WriteLine("The service is running. To stop press enter.");
    Console.ReadLine();

    // Detach the input channel and stop listening.
    // It releases the thread listening to messages.
    myReceiver.DetachDuplexInputChannel();
}

// It is called when a message is received.
private void OnMessageReceived(object sender, TypedRequestReceivedEventArgs<MyRequest> e)
{
    Console.WriteLine("Received: " + e.RequestMessage.Text);
```

4. 状态机保证状态转化的准确性，共五种状态两种操作，状态转化图如下



在状态机中实现动画开启暂停。

```
private void checkStateWithBtnClickEvent(int EVENT_BTN_CLICK)
{

    log("curr: " + currentState);
    switch (currentState) {
        case STATE_CLOSED:
            if (EVENT_BTN_CLICK == EVENT_BTN_CLOSE_CLICK)
            {
                //Do Nothing #CLOSED
                currentState = STATE_CLOSED;
                sbclosing.Stop();
                sbopening.Stop();
                mediaPlayer.Pause();
                mediaPlayer.Position = TimeSpan.Zero;
            }
            else if (EVENT_BTN_CLICK == EVENT_BTN_OPEN_CLICK)
            {
                //Transfer #OPENING
                currentState = STATE_OPENING;

                //#Closed -> # Opening
                sbclosing.Stop();
                sbopening.Begin();
                mediaPlayer.Position = TimeSpan.Zero;
                mediaPlayer.Play();
            }
            break;
        case STATE_CLOSING:
            if (EVENT_BTN_CLICK == EVENT_BTN_CLOSE_CLICK)
            {
                //Do nothing while closing
                currentState = STATE_CLOSING;
            }
            else if (EVENT_BTN_CLICK == EVENT_BTN_OPEN_CLICK)
            {
                //Pause it
                currentState = STATE_PAUSE;
```

5. COM3 接口的串口数据接收。直接采用 C#的 SerialPort 直接接收串口数据。同时将接收到的消息处理判断后输入状态机函数，实现状态迁移。同时也采用程序启动时开启子线程尝试连接 COM3 的串口，连接成功后持续监听该端口。

项目部署：

1. 先从 GitHub 上拉取 repository。
2. 导入 VS（推荐 VS2010），检查 reference 下是否导入了 Fleck 框架和 Eneter 框架（为保险起见，最好都导入），Fleck 可以采用 NuGet 安装，Eneter 采用手动 import 的方式即可。
3. 编译及打包发布
4. 注意：由于客户方要求，声音文件需多次变动，而打包发布后的程序会将声音文件一同打包编码，所以选用 hardcode 到路径：“d:\\test.mp3”为我们的目标音频，建议采用 10s 音频，不添加音频程序不报错，但播放动画时无声音。

Android 客户端：

功能：

1. 和服务端建立 Websocket 连接，并向其发送开启关闭的纯文本指令。

采用技术：

1. 采用 java_websocket 进行 Websocket 的建立和链接以及消息接受。
2. 设计逻辑回显错误以及服务链接状态显示。

关键技术介绍：

Android 比较简单，略。

项目部署：

1. 从 Github 上拉取 repository。
2. 导入 AndroidStudio。
3. 编译及打包发布加签。