

# 《人机交互原理与应用》实验报告

班级	信息十班	实验日期	2022.12.1	实验成绩	
姓名	陈澄	学号	32420212202930		
实验名称	人机交互原理与应用第六次实验				
实验目的、要求	<div>1、应用 unity 实现人脸识别。</div> <div>2、应用 unity 实现远程调用 API。</div> <div>3、应用 unity 实现各项权限的请求。</div>				
实验内容、步骤及结果	<div>实验内容：</div> <div>生成可执行程序（不限平台）并运行两个案例工程，在实验报告中<u>指出你的实验环境</u>。</div> <div>开发环境：Window 11 64 位，Unity 2021. 3. 10f1c1，VS Code</div> <div>运行环境：Window 11 64 位，使用电脑前置摄像头</div> <div>（Integrated Camera ）</div> <div>步骤：</div> <div>一、face++案例</div> <div>1、修改 lab6 工程内的 key，secret 为 face++平台获取的 key，secret。</div> <div>2、修改 Display 分辨率为自定义 1280*720。</div> <div>3、生成可运行程序并运行。</div>				

二、barracuda 案例

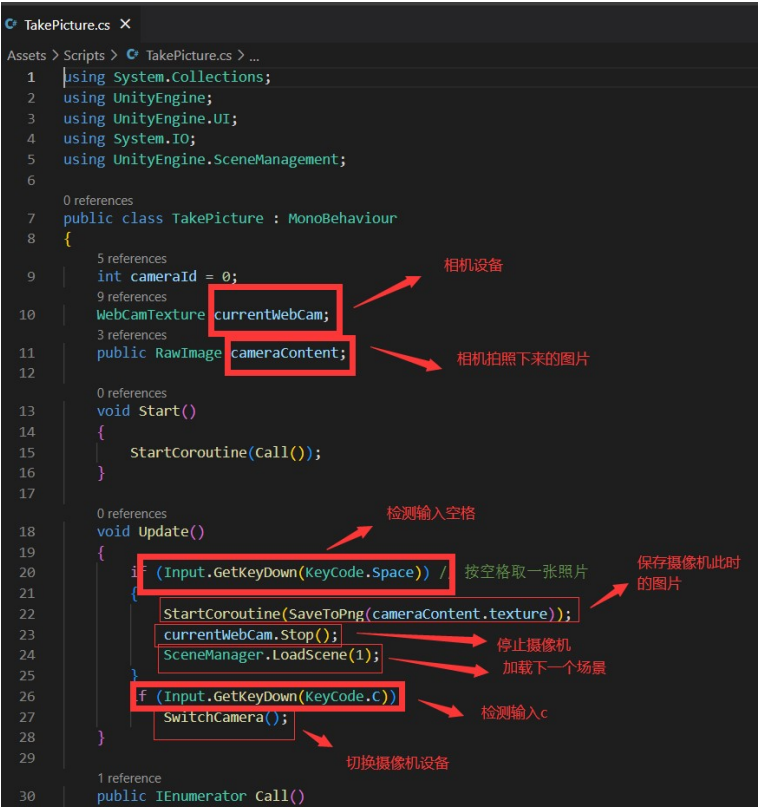
1、直接生成可执行程序

结果：

如演示视频所示

1. 从调用摄像头到获取人脸照片文件的过程；  
先等待键盘输入

回  
达  
相  
关  
问  
题



```
1 using System.Collections;
2 using UnityEngine;
3 using UnityEngine.UI;
4 using System.IO;
5 using UnityEngine.SceneManagement;
6
7 public class TakePicture : MonoBehaviour
8 {
9     5 references
10    int cameraId = 0;
11    9 references
12    WebCamTexture currentWebCam;
13    3 references
14    public RawImage cameraContent;
15
16    void Start()
17    {
18        StartCoroutine(Call());
19    }
20
21    void Update()
22    {
23        if (Input.GetKeyDown(KeyCode.Space)) / 按空格取一张照片
24        {
25            StartCoroutine(SaveToPng(cameraContent.texture));
26            currentWebCam.Stop();
27            SceneManager.LoadScene(1);
28        }
29        if (Input.GetKeyDown(KeyCode.C))
30        {
31            SwitchCamera();
32        }
33    }
34
35    public IEnumerator Call()
36    {
37    }
```

相机设备

相机拍照下来的图片

检测输入空格

按空格取一张照片

保存摄像机此时的图片

停止摄像机

加载下一个场景

检测输入c

切换摄像机设备

## 请求权限

```
1 reference
2 public IEnumerator Call()
3 {
4     // 请求权限
5     yield return Application.RequestUserAuthorization(UserAuthorization.WebCam);
6
7     if (Application.HasUserAuthorization(UserAuthorization.WebCam) && WebCamTexture.devices.Length > 0)
8     {
9         // 创建相机贴图
10        currentWebCam = new WebCamTexture(WebCamTexture.devices[cameraId].name, 1280, 720, 30);
11        cameraContent.texture = currentWebCam;
12        currentWebCam.Play();
13    }
14    else Debug.LogError("未获取摄像头权限");
15 }
```

请求用户的摄像头调用权限

若成功获得权限且找到设备

运行摄像头

设置摄像头显示图片分辨率

未获得权限或没有找到设备返回对应debug

## 保存图片

```
1 //保存png至StreamingAssets
2 1 reference
3 IEnumerator SaveToPng(Texture t)
4 {
5     yield return new WaitForEndOfFrame();
6     Texture2D png = new Texture2D(1280, 720, TextureFormat.ARGB32, false);
7     png.ReadPixels(new Rect(0, 0, 1280, 720), 0, 0);
8     png.Apply();
9     byte[] bytes = png.EncodeToPNG();
10    FileStream picFile = File.Open(Application.streamingAssetsPath + "/1.png", FileMode.Create);
11    BinaryWriter writer = new BinaryWriter(picFile);
12    writer.Write(bytes);
13    writer.Flush();
14    writer.Close();
15    picFile.Close();
16    Destroy(png);
17    png = null;
18    Debug.Log("图片保存成功, 文件名: 1.png");
19 }
20 }
```

创建铭文png的Texture2D变量并设置长

设置像素

转化为png形文件

二进制将该图片读入1.png

打开streamingasset目录并新建1.png文件

释放变量内存

打印相关debug

## 2. （Face++案例）如何调用人脸识别 API 获得人脸信息

```
1 //Face++ http请求
2 List<MultipartFormDataSection> formData = new List<MultipartFormDataSection>();
3 formData.Add(new MultipartFormDataSection("api_key", "p3me7a05yjd8t0p0c0s0MAC5Gv-c7X"));
4 formData.Add(new MultipartFormDataSection("api_secret", "tuh7M2xvKpy8-B00pn-ST-RFUbwe7qie"));
5 formData.Add(new MultipartFormDataSection("return_landmark", "1"));
6 string imgStr = Convert.ToBase64String(imgByte);
7 formData.Add(new MultipartFormDataSection("image_base64", imgStr));
8 formData.Add(new MultipartFormDataSection("return_attribute", "gender,age,emotion,beauty"));
9 UnityWebRequest www = UnityWebRequest.Post("https://api-cn.faceplusplus.com/facepp/v3/detect", formData);
10 yield return www.SendWebRequest();
11 }
```

调用API以及生成HTTP请求所需的所有信息，创建为formData变量

根据该变量访问相关网址并调用网站API

## 3. （Barracuda 案例）解释 RunInferenceBlazeFace 类中每个变量和方法的作用。

## 变量

```

1 using UnityEngine;
2 using UI = UnityEngine.UI;
3 using System.Collections.Generic;
4 using Unity.Barracuda;
5 using System.Runtime.InteropServices;
6 using UnityEngine.Android;
7
8 namespace MediaPipe.BlazeFace {
9
10     1 reference
11     public sealed class RunInferenceBlazeFace : MonoBehaviour
12     {
13         1 reference
14         private string _deviceName = ""; //设备名称
15         6 references
16         [SerializeField] Vector2Int _resolution = new Vector2Int(1080, 1080); //分辨率
17
18         12 references
19         WebCamTexture _webcam; //设备
20         4 references
21         RenderTexture _buffer; //渲染缓存
22
23         2 references
24         private Texture2D _image = null; //图片
25         1 reference
26         [SerializeField, Range(0, 1)] float _threshold = 0.75f; //阈值
27         2 references
28         [SerializeField] UI.RawImage _previewUI = null; //界面UI
29         1 reference
30         [SerializeField] Marker _markerPrefab = null; //标记
31
32         6 references
33         Marker[] _markers = new Marker[16]; //marker数组
34
35         // Maximum number of detections. This value must be matched with
36         // MAX_DETECTION in Common.hlsl.
37         2 references
38         const int MaxDetection = 64;
39
40     }
41
42     1 reference
43     public NNModel _model; //模型
44     1 reference
45     public ComputeShader _preprocess; //预处理器
46     1 reference
47     public ComputeShader _postprocess1; //预处理器1
48     1 reference
49     public ComputeShader _postprocess2; //预处理器2
50     5 references
51     ComputeBuffer _preBuffer; //预缓存
52     8 references
53     ComputeBuffer _post1Buffer; //缓存1
54     7 references
55     ComputeBuffer _post2Buffer; //缓存2
56     7 references
57     ComputeBuffer _countBuffer; //缓存3
58     10 references
59     IWorker _worker; //IWorker
60     9 references
61     int _size; //内存大小
62
63     0 references
64     public GameObject webglWarning; //报错
65     4 references
66     public UI.Dropdown _cameraDropdown; //选择相机设备UI
67     8 references
68     public UI.Dropdown _backendDropdown; //选择预编译器UI
69
70     //
71     // Detection structure. The layout of this structure must be matched with
72     // the one defined in Common.hlsl.
73     //
74     [StructLayout(LayoutKind.Sequential)]
75     7 references
76     public readonly struct Detection
77     {
78         // Bounding box
79         1 reference
80         public readonly Vector2 center; //中心点位置向量

```

```

49 public readonly struct Detection
50 {
51     // Bounding box
52     1 reference
53     public readonly Vector2 center; //中心点位置向量
54     1 reference
55     public readonly Vector2 extent; //面积大小向量
56     // Key points
57     1 reference
58     public readonly Vector2 leftEye; //左眼位置向量
59     1 reference
60     public readonly Vector2 rightEye; //右眼位置向量
61     1 reference
62     public readonly Vector2 nose; //鼻子位置向量
63     1 reference
64     public readonly Vector2 mouth; //嘴位置向量
65     1 reference
66     public readonly Vector2 leftEar; //左耳位置向量
67     1 reference
68     public readonly Vector2 rightEar; //右耳位置向量
69     // Confidence score [0, 1]
70     1 reference
71     public readonly float score; //相似度
72     // Padding
73     0 references | 0 references | 0 references
74     public readonly float pad1, pad2, pad3; //图像矩形尺寸
75     // sizeof(Detection)
76     2 references
77     public const int Size = 20 * sizeof(float); //数据占用内存大小
78 };

```

## 方法

```

73 0 references
74 void Start() //初始化
75 {
76     // Prepare camera
77     #if PLATFORM_ANDROID
78     Permission.RequestUserPermission(Permission.Camera);
79     while (!Permission.HasUserAuthorizedPermission(Permission.Camera))
80     {
81     } //安卓端获得摄像头权限
82     #endif
83     #if UNITY_IOS
84     Application.RequestUserAuthorization(UserAuthorization.WebCam);
85     while (!Application.HasUserAuthorization(UserAuthorization.WebCam))
86     {
87     } //ios端获得摄像头权限
88     #endif
89     #if UNITY_WEBGL
90     WebGLWarning.SetActive(true);
91     #endif
92     _webcam = new WebCamTexture(_deviceName, _resolution.x, _resolution.y);
93     _buffer = new RenderTexture(_resolution.x, _resolution.y, 0);
94     _webcam.requestedFPS = 30; //设置帧率
95     _webcam.Play(); //运行摄像机
96     Screen.orientation = ScreenOrientation.LandscapeLeft; //设置为强制不旋转横屏
97     AddCameraOptions(); //增加摄像机设备选择
98     AddBackendOptions(); //增加预编译器选择
99     //initialization
100     AllocateObjects(); //初始化游戏对象

```

```

102     AddBackendOptions(); // 增加后端编译器选择
103
104     // initialization
105     AllocateObjects(); // 初始化游戏对象
106
107     // Marker population
108     for (var i = 0; i < _markers.Length; i++)
109     {
110         _markers[i] = Instantiate(_markerPrefab, _previewUI.transform, Quaternion.identity);
111     }
112     // Static image test: Run the detector once.
113     if (_image != null) runInference(_image);
114 }
115
116 0 references
117 void Update() // 格式化输入
118 {
119     // Format video input
120     if (!_webcam.didUpdateThisFrame) return;
121
122     var aspect1 = (float)_webcam.width / _webcam.height;
123     var aspect2 = (float)_resolution.x / _resolution.y;
124     var gap = aspect2 / aspect1;
125
126     var vflip = _webcam.videoVerticallyMirrored;
127     var scale = new Vector2(gap, vflip ? -1 : 1);
128     var offset = new Vector2((1 - gap) / 2, vflip ? 1 : 0);
129
130     Graphics.Blit(_webcam, _buffer, scale, offset);
131 }
132
133 0 references
134 void LateUpdate() // 运行判断框架
135 {
136     // Webcam test: Run the detector every frame.
137     runInference(_buffer);
138 }

```

```

137 1 reference
138 public void AddCameraOptions() // 增加摄像机设备选择
139 {
140     List<string> options = new List<string> ();
141     foreach (var option in WebCamTexture.devices) {
142         options.Add(option.name);
143     }
144     _cameraDropdown.ClearOptions ();
145     _cameraDropdown.AddOptions(options);
146 }
147
148 1 reference
149 public void AddBackendOptions() // 增加预编译器选择
150 {
151     List<string> options = new List<string> ();
152     #if !UNITY_WEBGL
153     options.Add("ComputePrecompiled");
154     #endif
155     //options.Add("PixelShader"); //not supported with this demo
156     //options.Add("CSharpBurst"); //not supported with this demo
157     _backendDropdown.ClearOptions();
158     _backendDropdown.AddOptions(options);
159 }
160
161 0 references
162 public void SwapCamera() // 切换摄像头
163 {
164     _webcam.Stop();
165     _webcam.deviceName = _cameraDropdown.options[_cameraDropdown.value].text;
166     _webcam.Play();
167 }
168
169 1 reference
170 public void ProcessImage(Texture image, float threshold = 0.75f) // 处理图像
171 {
172     => ExecuteML(image, threshold);
173 }
174
175 1 reference
176 public void AllocateObjects() // 初始化游戏对象

```



```

196     }
197 }
198
199 1 reference
void ExecuteML(Texture source, float threshold)//处理缓存, 模型图像等信息
{
200     // Reset the compute buffer counters.
201     _post1Buffer.SetCounterValue(0);
202     _post2Buffer.SetCounterValue(0);
203
204     // Preprocessing
205     var pre = _preprocess;
206     pre.SetInt("_ImageSize", _size);
207     pre.SetTexture(0, "Texture", source);
208     pre.SetBuffer(0, "_Tensor", _preBuffer);
209     pre.Dispatch(0, _size / 8, _size / 8, 1);
210
211     // Run the BlazeFace model.
212     using (var tensor = new Tensor(1, _size, _size, 3, _preBuffer))
213     {
214         _worker.Execute(tensor);
215     }
216
217     // Output tensors -> Temporary render textures
218     var scores1RT = _worker.CopyOutputToTempRT("Identity", 1, 512);
219     var scores2RT = _worker.CopyOutputToTempRT("Identity_1", 1, 384);
220     var boxes1RT = _worker.CopyOutputToTempRT("Identity_2", 16, 512);
221     var boxes2RT = _worker.CopyOutputToTempRT("Identity_3", 16, 384);
222
223     // 1st postprocess (bounding box aggregation)
224     var post1 = _postprocess1;
225     post1.SetFloat("_ImageSize", _size);
226     post1.SetFloat("_Threshold", threshold);
227
228     post1.SetTexture(0, "_Scores", scores1RT);
229     post1.SetTexture(0, "_Boxes", boxes1RT);
230     post1.SetBuffer(0, "_Output", _post1Buffer);
231     post1.Dispatch(0, 1, 1, 1);
232 }

```

```

233
234 2 references
void runInference(Texture input)//进行人脸识别
{
235     // Face detection
236     ProcessImage(input, _threshold);
237
238     // Marker update
239     var i = 0;
240
241     foreach (var detection in Detections)
242     {
243         if (i == _markers.Length) break;
244         var marker = _markers[i++];
245         marker.detection = detection;
246         marker.gameObject.SetActive(true);
247     }
248
249     for (; i < _markers.Length; i++)
250     {
251         _markers[i].gameObject.SetActive(false);
252     }
253
254     // UI update
255     _previewUI.texture = input;
256 }
257
258 1 reference
public IEnumerable<Detection> Detections
259     => _post2ReadCache ?? UpdatePost2ReadCache();
260
261 5 references
Detection[] _post2ReadCache;
262
263 2 references
int[] _countReadCache = new int[1];
264
265 1 reference
Detection[] UpdatePost2ReadCache()
266 {

```

```

294         _post2ReadCache = new Detection[count];
295         _post2Buffer.GetData(_post2ReadCache, 0, 0, count);
296
297         return _post2ReadCache;
298     }
299
300     0 references
301     void OnDestroy()//销毁各方面数据释放内存
302     {
303         Destroy(_webcam);
304         Destroy(_buffer);
305         Dispose();
306     }
307
308     public void Dispose()//调用销毁缓存函数
309         => DeallocateObjects();
310
311     1 reference
312     void DeallocateObjects()//销毁缓存
313     {
314         _preBuffer?.Dispose();
315         _preBuffer = null;
316
317         _post1Buffer?.Dispose();
318         _post1Buffer = null;
319
320         _post2Buffer?.Dispose();
321         _post2Buffer = null;
322
323         _countBuffer?.Dispose();
324         _countBuffer = null;
325
326         _worker?.Dispose();
327         _worker = null;
328     }

```