Приложение 5 - Извадка от кода за динамичната функционалност на C#

1.DynamicTargetLoader.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Vuforia;

public class DynamicTargetLoader : MonoBehaviour {

    // specify these in Unity Inspector
    private  const string DataSetName = "ARMathDB";  //
Assets/StreamingAssets/QCAR/DataSetName



    // Use this for initialization
    void Start () {
        VuforiaARController.Instance.RegisterVuforiaStartedCallback(LoadDataSet);
    }

    // Update is called once per frame
    void Update () {

       }




    void LoadDataSet()
    {

        ObjectTracker objectTracker = TrackerManager.Instance.GetTracker<ObjectTracker>();
        DataSet dataSet = objectTracker.CreateDataSet();

        if (dataSet.Load(DataSetName))
        {

            objectTracker.Stop();  // stop tracker so that we can add new dataset

            if (!objectTracker.ActivateDataSet(dataSet))
            {
                Debug.Log("<color=yellow>Failed to Activate DataSet: " + DataSetName +
"</color>");
            }

            if (!objectTracker.Start())
            {
                Debug.Log("<color=yellow>Tracker Failed to Start.</color>");
            }

            int counter = 0;

            IEnumerable<TrackableBehaviour> allTrackableBehaviours =
TrackerManager.Instance.GetStateManager().GetTrackableBehaviours();
            foreach (TrackableBehaviour currentTrackableBehaviour in allTrackableBehaviours)
            {
                if (currentTrackableBehaviour.name == "New Game Object")
                {

                    // change generic name to include trackable name
```

```
                    currentTrackableBehaviour.gameObject.name = ++counter +
":DynamicImageTarget-" + currentTrackableBehaviour.TrackableName;

                    // add additional script components for trackable

currentTrackableBehaviour.gameObject.AddComponent<DefaultTrackableEventHandler>();
                    currentTrackableBehaviour.gameObject.AddComponent<TurnOffBehaviour>();

                                    //attach the custom event handler to be invoked when
tracker image is detected or lost
                                    ITrackableEventHandler handler = new
ARMathTrackableEventHandler(currentTrackableBehaviour);
                    currentTrackableBehaviour.RegisterTrackableEventHandler(handler);
                }
            }
        }
        else
        {
            Debug.LogError("<color=yellow>Failed to load dataset: '" + DataSetName +
"'</color>");
        }
    }
}
```

---

2. ARMathTrackableEventHandler.cs

```
using UnityEngine;
using System.Collections;
using Vuforia;
using System;
using Lean.Touch;

public class ARMathTrackableEventHandler : ITrackableEventHandler
{
    private TrackableBehaviour mTrackableBehaviour;
    private GameObject mModelObject;
    private ARMathMainController mGuiManagement;
    public ARMathTrackableEventHandler(TrackableBehaviour trackableBehaviour)
    {
              // store instance of the TrackableBehaviour this handler is associated to
              // it is needed later, when events are detected
        mTrackableBehaviour = trackableBehaviour;
        mGuiManagement =
GameObject.Find("MainController").GetComponent<ARMathMainController>();
    }

    public void OnTrackableStateChanged(
          TrackableBehaviour.Status previousStatus,
          TrackableBehaviour.Status newStatus)
    {

        Debug.Log("OnTrackableStateChanged old: " + previousStatus + " new: " + newStatus);
        if (newStatus == TrackableBehaviour.Status.DETECTED || newStatus ==
TrackableBehaviour.Status.TRACKED)
        {
            OnTrackingFound();

        } else if (   newStatus != TrackableBehaviour.Status.EXTENDED_TRACKED) // in case
state changes to EXTENDED_TRACKED no need to do anything
        {
            OnTrackingLost();
        }
```

```
    }

    private void OnTrackingFound()
    {
        Debug.Log("Trackable detected: " + mTrackableBehaviour.TrackableName);
        mGuiManagement.ShowButtons();
        mGuiManagement.ActivateTrackable(mTrackableBehaviour);
    }

    private void OnTrackingLost()
    {
        Debug.Log("Trackable lost: " + mTrackableBehaviour.TrackableName);
        mGuiManagement.HideButtons();
        mGuiManagement.DeactivateTrackable();
    }

}
```

---

3. ARMathMainController.cs

```
using Lean.Touch;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Vuforia;

public class ARMathMainController : MonoBehaviour {

    private bool mShowGUIButton = false;
    private GameObject mGuiCanvas = null;
    private GameObject mActiveModel = null;
    private TrackableBehaviour mActiveTrackable=null;
    private bool mRotate = false;
    private int mRotateDirection = -1;
    private int mCurrentModelIndex = 0;



    // Use this for initialization
    void Start () {


    }

       // Update is called once per frame
       void Update () {

       }

    // Executed by Unity on fixed time intervals, independent from frame rendering
    void FixedUpdate()
    {
        if(mActiveModel!=null && mRotate)
        {
            Vector3 rotationVector = mRotateDirection == -1 ? Vector3.up : Vector3.down;
            mActiveModel.transform.Rotate(rotationVector);
        }
    }

    // invoked when trackable image is detected
    public void ShowButtons()
```

```
    {
        mShowGUIButton = true;
        GameObject canvas = FindCanvas();
        canvas.SetActive(true);

    }

    // invoked when trackable image is lost
    public void HideButtons()
    {
        mShowGUIButton = false;
        GameObject canvas =FindCanvas();
        canvas.SetActive(false);
    }


    private GameObject FindCanvas()
    {
           // Search for the canvas only the first time, then store the reference for future
use.
        if (mGuiCanvas == null)
        {
            mGuiCanvas = GameObject.Find("Canvas");
            Debug.Log("Canvas found: " + mGuiCanvas);
        }
        return mGuiCanvas;
    }


    public void StartRotateModel(int rotationDirection)
    {
       // Commented code is for demo purposes. If uncommented it will make screenshot every
time "rotate" button is pressed.
       // Screenshots will be located in Android\data\com.unisofia.ARMath3\files on the
external SD Card
          // string timeStamp = System.DateTime.Now.ToString("HHmmssffff");
       // Debug.Log("ARMathScreenshot" + timeStamp + ".png");
       // Application.CaptureScreenshot("ARMathScreenshot"+timeStamp+".png");
          //------------------------
        mRotateDirection = rotationDirection;
        mRotate = true;
        Debug.Log("mActiveModel=" + mActiveModel);
    }

    public void StopRotateModel()
    {
        mRotate = false;
    }

    // method used by the GUI Button "||>"
    public void ShowNextModel()
    {
        //increase the index
        mCurrentModelIndex++;
              //try to find a model with the new index
        GameObject augmentationPrefab =
loadAugmentationPrefab(mActiveTrackable.TrackableName, mCurrentModelIndex); ;
        if (augmentationPrefab != null)
        {
            SwapAugmentation(mActiveTrackable, augmentationPrefab);

        } else
        {   // if model with the current index is not found, rotate the index back to 0 and
load the first model
```

```
            mCurrentModelIndex = 0;
            GameObject firstAugmentationPrefab =
loadAugmentationPrefab(mActiveTrackable.TrackableName, mCurrentModelIndex); ;
            if (firstAugmentationPrefab != null)
            {
                SwapAugmentation(mActiveTrackable, firstAugmentationPrefab);

            }
        }
    }

       // method used by the GUI Button "<||"
    public void ShowPreviousModel()
    {
           // if this is the first model - do nothing
        if (mCurrentModelIndex > 0)
        {
            mCurrentModelIndex--;
            GameObject augmentationPrefab =
loadAugmentationPrefab(mActiveTrackable.TrackableName, mCurrentModelIndex); ;
            if (augmentationPrefab != null)
            {
                SwapAugmentation(mActiveTrackable, augmentationPrefab);

            }
        }

    }

    public void SetActiveModel(GameObject modelObject)
    {
        mActiveModel = modelObject;
    }

    public void SetActiveTrackable(TrackableBehaviour trackable)
    {
        mActiveTrackable = trackable;
    }


    public void DeactivateTrackable()
    {
        if (mActiveModel != null)
        {
            mActiveModel.SetActive(false);
        }
        mActiveModel = null;
        mActiveTrackable = null;
    }

    public void ActivateTrackable(TrackableBehaviour trackable)
    {
        mActiveTrackable = trackable;
        Debug.Log("Trackable detected: " + mActiveTrackable.TrackableName);

        // augmentationObject = GameObject.CreatePrimitive(PrimitiveType.Cube);
        // instantiate augmentation object and swap the current visualization
        GameObject augmentationPrefab =
loadAugmentationPrefab(mActiveTrackable.TrackableName, mCurrentModelIndex);
        SwapAugmentation(mActiveTrackable, augmentationPrefab);


    }
```

```
    private GameObject loadAugmentationPrefab(string prefab_name, int prefab_index)
    {
            // Dynamically load a 3d model from the Resources folder
        Debug.Log("Loading prefab for " + prefab_name + " with index:" + prefab_index);
        GameObject augmentationPrefab = (GameObject)Resources.Load(prefab_name + "_" +
prefab_index);
        Debug.Log("Prefab loaded: " + augmentationPrefab);
        return augmentationPrefab;
    }

    private void SwapAugmentation(TrackableBehaviour trackableBehaviour, GameObject
augmentationPrefab)
    {
        GameObject trackableGameObject = trackableBehaviour.gameObject;
        //disable any pre-existing augmentation
        for (int i = 0; i < trackableGameObject.transform.childCount; i++)
        {
            Transform child = trackableGameObject.transform.GetChild(i);
            child.gameObject.SetActive(false);
        }

        if (augmentationPrefab != null)
        {
                // create new instance of the loaded object
            GameObject augmentation =
(GameObject)GameObject.Instantiate(augmentationPrefab);//GameObject.CreatePrimitive(Primitiv
eType.Cube); //(GameObject)GameObject.Instantiate(augmentationObject);
            // put it under the Trackable in the scene hierarchy. This is the way to
associate the object with a marker image.
                    augmentation.transform.parent = mActiveTrackable.gameObject.transform;
                    //set additional properties of the new object
            augmentation.transform.localPosition = new Vector3(0f, 0f, 0f);
            augmentation.transform.localRotation = Quaternion.identity;
            augmentation.transform.localScale = new Vector3(1.0f, 1.0f, 1.0f);
            // attach scripts for scale and rotate with multi-touch gestures.
                    InitTouchControllerWithObject(augmentation);
                    //activate the new object
            augmentation.gameObject.SetActive(true);
            mActiveModel = augmentation;

        }
    }

    private void InitTouchControllerWithObject(GameObject augmentation)
    {
        // attach script for scaling with "pinch zoom" gesture
        augmentation.AddComponent(typeof(ARMathLeanScale));
              // attach script for rotation with two finger gesture
        augmentation.AddComponent(typeof(ARMathLeanRotate));
    }
}
```

4.ARMathLeanScale.cs

```
using Lean.Touch;
using System.Collections.Generic;
using UnityEngine;


// This script allows you to scale the current augmentation model
```

```
public class ARMathLeanScale : MonoBehaviour
       {

        private const int RequiredFingerCount = 0;

        // sensitivity for the mouse wheel when simulating pinching
              public const float WheelSensitivity = 0;


              protected virtual void Update()
              {
                     // Get the fingers used
                   List<LeanFinger> fingers = LeanTouch.GetFingers(true,
RequiredFingerCount);

                     // Calculate the scaling values based on the  fingers
                     float scale        = LeanGesture.GetPinchScale(fingers,
WheelSensitivity);

                     // Perform the scaling
                     Scale(scale);
              }

              private void Scale(float scale)
              {

                     if (scale > 0.0f)
                     {
                            // Grow the local scale by scale
                            transform.localScale *= scale;

                     }
              }
       }
```

5.ARMathLeanRotate.cs

```
using Lean.Touch;
using System.Collections.Generic;
using UnityEngine;


       // This script allows you to transform the current augmentation model
       public class ARMathLeanRotate : MonoBehaviour
       {

              private const int RequiredFingerCount=0;

              private  Vector3 RotateAxis = Vector3.up;

              protected virtual void Update()
              {
                     // Get the fingers used. First argument specifies, that fingers on GUI
elements will be ignored.
                     List<LeanFinger>  fingers = LeanTouch.GetFingers(true,
RequiredFingerCount);

                     // Calculate the rotation values based on the fingers
                     float  degrees = LeanGesture.GetTwistDegrees(fingers);
```

```
                // Perform the rotation
                Rotate( degrees);
        }

        private void Rotate( float degrees)
        {

                transform.rotation *= Quaternion.AngleAxis(degrees, RotateAxis);

        }
    }
```