```
//Матеріали взяв з сайту           https://null-code.ru/scripts/115-
prostoy-primer-risovaniya-v-unity.html і доопрацював
//На сцені у нас дві камери, перша, дивиться на «чистий аркуш»,
//тобто об'єкт з матеріалом, але без текстури, так само, перед цією
камерою створюються клони кисті, з яких утворюється малюнок,
//а камера в свою чергу передає зображення в рендер текстуру. Ця текстура
причеплено на матеріал іншого об'єкта, який знаходиться перед другою
камерою.
//Тобто коли ми водимо мишкою по полотну(де рендер текстура), то
створюємо клони пензлика там, де знаходиться перша камера, а вона
повертає нам текстуру з малюнком.
//Скрипт вішаємо на головну камеру
using UnityEngine;
using System.Collections.Generic;
public class Paint : MonoBehaviour
{
    private Animator animator;
    private GameObject Player;

    public SpriteRenderer kistochka; // спрайт кісточкі
      public Color colorKistochky = Color.red;//Колір кісточки
      [Range(0.1f, 0.2f)] public float sizeKistochky = 0.1f;//Розмір
кісточки
      public Camera cameraRT; // дублююча камера
      private int sizeRT = 1024; // Зорзмір текстури
      public MeshRenderer canvasObject; // Головний Quad по якому і
будемо малювати
      public MeshRenderer planeRT; // дуюлюючий Quad на якому
відображається малювання

    private RenderTexture renderTexture;
      private Vector3 position;

    //public float _SizeBox=0.1f;
    public float _SizeSphere = 0.2f;

    private GameObject _feet_left;
    private GameObject _feet_right;
    private GameObject _mesh;
    private MeshFilter _mesh_f;
    private SphereCollider _mesh_b;

    private GameObject[] _clone;

    private Vector3 L1;
    private Vector3 L2;
    private Vector3 L3;

    private int counter = 0, maxCount = 1000; // счетчик и макс.
количество объектов
      private bool isSave;
    public static List<Transform> positionCreate = new
List<Transform>();//масив для точок по яким малювати
    private Vector3[] _vertices = new Vector3[1000];//масив точок
    void Awake()
      {
            position.z = kistochka.transform.position.z;//стала позиція по
Z
            Clear();
      }
    private void Start()
```

```
    {
        _feet_left = GameObject.Find("TTBFeetB_Left") as GameObject;
        _feet_right = GameObject.Find("TTBFeetB_Right") as GameObject;
        _mesh = GameObject.Find("Mesh") as GameObject;
        _mesh_f = GameObject.Find("Mesh").GetComponent<MeshFilter>();
        _mesh_b = GameObject.Find("Mesh").GetComponent<SphereCollider>();


        Player = GameObject.Find("TTBoyB");
        animator = Player.GetComponentInChildren<Animator>();
    }

    void Update()
      {

        var animatorStateInfo = animator.GetCurrentAnimatorStateInfo(0);
        if (Input.GetMouseButton(0) && !isSave)//якщо ЛКМ нажата і не
було збережено
            {

             if (animatorStateInfo.IsName("TTB_walk2") ||
animatorStateInfo.IsName("TTB_walk2 0"))
             {
                 //Debug.Log("Done anim");
                 animator.speed = 0.5f;//зповільнюємо анімацію
                 Mov.speedPlayer = Mov.speedPlayer/2f;//зменшуємо
швидкість руху
                // Player.transform.position = new
Vector3(Player.transform.position.x, Player.transform.position.y + 0.5f,
Player.transform.position.z);
             }
             kistochka.gameObject.SetActive(true);

             Draw();//малювання
            }
            else
        if(Input.GetMouseButtonUp(0))//Якщо ЛКМ віджата
            {
             animator.speed = 1f;//Повертаємо швидкість анімації в норму
             Mov.speedPlayer = 14f;//Повертаємо швидкість руху в норму
             Save();
             //Clear();//очистка
        }
            else
        if(isSave)//якщо щось намалювали можна зберегти але оскільки
очищаєм при віджиманні кнопки то сюди доходити не буде
            {
             animator.speed = 1f;//Повертаємо швидкість анімації в норму
             Mov.speedPlayer = 14f;//Повертаємо швидкість руху в норму
             //Save();
             kistochka.gameObject.SetActive(false);//Кісточку робимо не
активною, щоб можна було активувати при наступному мазку, і щоб не
заважала

////////////////////////////////////////////////////////////////////
             _clone = GameObject.FindGameObjectsWithTag("Clone");
             if (_clone.Length > 0)
             {
                 Debug.Log($"Dedstroyed{_clone.Length}");
                 foreach (GameObject ds in _clone)
                 {
```

```
                    Destroy(ds);
                }
            }
            else
                return;

//////////////////////////////////////////////////////////////////
            Clear();
        }
           else //Якщо нічого не намальовано
           {
            animator.speed = 1f;//Повертаємо швидкість анімації в норму
            Mov.speedPlayer = 14f;//Повертаємо швидкість руху в норму
            kistochka.gameObject.SetActive(false);//кісточка не активна
(Brush)
           }



    }
      void Save() //Метод для збереження (але оскільки нам зберегтии
треба лише координати то це можна буде зробити і перед очисткою)
      {
        //Debug.Log("Save Done");
        counter = 0;//Обнуляємо оюмеження вводу
        _vertices = null;

        RenderTexture.active = renderTexture;
           Texture2D tex = new Texture2D(renderTexture.width,
renderTexture.height, TextureFormat.RGB24, false);//в змінну tex
записуємо параметри зображення
        tex.ReadPixels(new Rect (0, 0, renderTexture.width,
renderTexture.height), 0, 0);//зчитати данні з зображення
           tex.Apply();//Фактично застосувати всі попередні зміни
SetPixel та SetPixels.
        RenderTexture.active = null;
           planeRT.material.mainTexture = tex;//присвоюємо зчитані данні
зображення до дублюючого Quad

//////////////////////////////////////////////////////////////////////
///////////////вивільнення памяті
        foreach (Transform child in planeRT.transform)
        {
            positionCreate.Add(child);
            Destroy(child.gameObject);
        }



//////////////////////////////////////////////////////////////////////
//////////////////Перевірка заповнення списка
        //if (positionCreate.Count != 0)
        //{
        //    foreach (Transform ch in positionCreate)
        //    {
        //        Debug.Log($"Take=\t{ch.transform.position}");
        //    }
        //}

//////////////////////////////////////////////////////////////////////
///////////////
```

```
        isSave = false;//обнуляємо збереження
        Clear();
      }
      void Clear() // очищення
      {
        //Debug.Log("Clear Done");
        counter = 0;//Обнуляємо оюмеження вводу
        _vertices = null;

////////////////////////////////////////////////////////////////////////
///////////////Записуємо в список і видаляємо
        if (planeRT.gameObject.activeSelf)
        foreach (Transform child in planeRT.transform)
            {
             Destroy(child.gameObject);
            }
        Destroy(planeRT.material.mainTexture);
            Destroy(renderTexture);
            renderTexture = new RenderTexture(sizeRT, sizeRT, 24,
RenderTextureFormat.ARGB32);
            cameraRT.targetTexture = renderTexture;
            canvasObject.material.mainTexture = renderTexture;
        //positionCreate.Clear();

    }

      void Draw() // малювання (клонування кісточки(Brush))
      {
            RaycastHit hit;
            Ray ray =
Camera.main.ScreenPointToRay(Input.mousePosition);//Зчитуємо з екрану
координати курсору
            if(Physics.Raycast(ray, out hit))
            {
                  SpriteRenderer s = Instantiate(kistochka) as
SpriteRenderer;//Створення дублікату кісточки як спрайту



             Vector2 uv = new Vector2(hit.textureCoord.x,
hit.textureCoord.y);//Координата текстури УФ в місці зіткнення.
             position.x = uv.x - cameraRT.orthographicSize;//Напіврозмір
камери в ортографічному режимі.
             position.y = uv.y - cameraRT.orthographicSize;
             s.color = colorKistochky;
                  s.transform.localPosition = position;
                  s.transform.localScale = Vector3.one * sizeKistochky;
                  s.transform.parent = planeRT.transform;
                  counter++;
                  if(counter > maxCount)
                  {
                        isSave = true;//Переключитись на збереження
                  }

                 //L1 = new Vector3(0f, 0f, _SizeBox);
                 //L2 = new Vector3(0f, _SizeBox, 0f);
                 //L3 = new Vector3(_SizeBox, 0f, 0f);

             //_mesh_b.size = new Vector3(Mathf.Abs(position.x),
Mathf.Abs(position.y), _SizeBox);
```

```
            _mesh_b.radius =_SizeSphere;
            //_mesh_f.mesh = Cube(L1, L2, L3);
            _mesh_f.mesh = Icosahedron(0.2f);//
            GameObject _leftHand = Instantiate(_mesh, new
Vector3(_feet_left.transform.position.x + position.x,
_feet_left.transform.position.y + position.y-0.2f,
_feet_left.transform.position.z + 0.2f), Quaternion.identity) as
GameObject;
            _leftHand.transform.parent = _feet_left.transform;
            _leftHand.tag = "Clone";
            GameObject _rightHand = Instantiate(_mesh, new
Vector3(_feet_right.transform.position.x + position.x,
_feet_right.transform.position.y + position.y - 0.2f,
_feet_right.transform.position.z - 0.2f), Quaternion.identity) as
GameObject;
            _rightHand.transform.parent = _feet_right.transform;
            _rightHand.tag = "Clone";




        }
    }
    /// <summary>
    ///
////////////////////////////////////////////////////////////////////////
////////////////////
    /// </summary>
    /// <param name="width"></param>
    /// <param name="length"></param>
    /// <param name="height"></param>
    /// <returns></returns>
    ///



    public static Mesh Icosahedron(float radius)//?косаЇдр
    {
        var magicAngle = Mathf.PI * 26.565f / 180;
        var segmentAngle = Mathf.PI * 72 / 180;
        var currentAngle = 0f;

        var v = new Vector3[12];
        v[0] = new Vector3(0, radius, 0);
        v[11] = new Vector3(0, -radius, 0);

        for (var i = 1; i < 6; i++)
        {
            v[i] = new Vector3(radius * Mathf.Sin(currentAngle) *
Mathf.Cos(magicAngle),
                radius * Mathf.Sin(magicAngle),
                radius * Mathf.Cos(currentAngle) *
Mathf.Cos(magicAngle));
            currentAngle += segmentAngle;
        }
        currentAngle = Mathf.PI * 36 / 180;
        for (var i = 6; i < 11; i++)
        {
            v[i] = new Vector3(radius * Mathf.Sin(currentAngle) *
Mathf.Cos(-magicAngle),
                radius * Mathf.Sin(-magicAngle),
```

```
                radius * Mathf.Cos(currentAngle) * Mathf.Cos(-
magicAngle));
            currentAngle += segmentAngle;
        }

        var combine = new CombineInstance[20];
        combine[0].mesh = Triangle(v[0], v[1], v[2]);
        combine[1].mesh = Triangle(v[0], v[2], v[3]);
        combine[2].mesh = Triangle(v[0], v[3], v[4]);
        combine[3].mesh = Triangle(v[0], v[4], v[5]);
        combine[4].mesh = Triangle(v[0], v[5], v[1]);

        combine[5].mesh = Triangle(v[11], v[7], v[6]);
        combine[6].mesh = Triangle(v[11], v[8], v[7]);
        combine[7].mesh = Triangle(v[11], v[9], v[8]);
        combine[8].mesh = Triangle(v[11], v[10], v[9]);
        combine[9].mesh = Triangle(v[11], v[6], v[10]);

        combine[10].mesh = Triangle(v[2], v[1], v[6]);
        combine[11].mesh = Triangle(v[3], v[2], v[7]);
        combine[12].mesh = Triangle(v[4], v[3], v[8]);
        combine[13].mesh = Triangle(v[5], v[4], v[9]);
        combine[14].mesh = Triangle(v[1], v[5], v[10]);

        combine[15].mesh = Triangle(v[6], v[7], v[2]);
        combine[16].mesh = Triangle(v[7], v[8], v[3]);
        combine[17].mesh = Triangle(v[8], v[9], v[4]);
        combine[18].mesh = Triangle(v[9], v[10], v[5]);
        combine[19].mesh = Triangle(v[10], v[6], v[1]);

        var mesh = new Mesh();
        mesh.CombineMeshes(combine, true, false);
        return mesh;
    }
    public static Mesh Triangle(Vector3 vertex0, Vector3 vertex1, Vector3
vertex2)
    {
        var normal = Vector3.Cross((vertex1 - vertex0), (vertex2 -
vertex0)).normalized;
        var mesh = new Mesh
        {
            vertices = new[] { vertex0, vertex1, vertex2 },
            normals = new[] { normal, normal, normal },
            uv = new[] { new Vector2(0, 0), new Vector2(0, 1), new
Vector2(1, 1) },
            triangles = new[] { 0, 1, 2 }
        };
        return mesh;
    }









    public static Mesh Cube(Vector3 width, Vector3 length, Vector3
height)
    {
```

```
       // Paint pain = new Paint();

           var corner0 = -width / 2 - length / 2 - height / 2;
        var corner1 = width / 2 + length / 2 + height / 2;

        var combine = new CombineInstance[6];
        combine[0].mesh = Quad(corner0, length, width );//лева стенка
вверх вперед
        combine[1].mesh = Quad(corner0, width, height);//нижня стєнка
вперед в право
        combine[2].mesh = Quad(corner0, height , length );//передня
стєнка вправо вверх
        combine[3].mesh = Quad(corner1, -width , -length);//права стєнка
до мене вниз
        combine[4].mesh = Quad(corner1, -height , -width);//верхня стєнка
вліво
        combine[5].mesh = Quad(corner1, -length, -height);//задня стєнка
вниз

        var mesh = new Mesh();
        mesh.CombineMeshes(combine, true, false);
        return mesh;
    }
    public static Mesh Quad(Vector3 origin, Vector3 width, Vector3
length)
    {
        var normal = Vector3.Cross(length, width).normalized;
        var mesh = new Mesh
        {
            vertices = new[] { origin, origin + length, origin + length +
width, origin + width },
            normals = new[] { normal, normal, normal, normal },
            uv = new[] { new Vector2(0, 0), new Vector2(0, 1), new
Vector2(1, 1), new Vector2(1, 0) },
            triangles = new[] { 0, 1, 2, 0, 2, 3 }
        };
        return mesh;
    }
}
```