ABpainting—a rhythm game with artwork based on Unity3D

Developer intent

I wanted to imitate a rhythm game like OSU long time ago and finally I could realize it on Unity by chance. I combine the rhythm game and painting on background together to make an innovation. Though the product turns out imperfect, I will keep improving the codes for a better game experience.

Scope

C# language Unity3D engine MonoDevelop IDE

User guide

Touch or click the beat (circle) at the right time--seeing the white ring become smaller and almost contact the red circle. If the player misses the right time, the HP value will decrease. You can recover your HP every time you catch the right time. Stay alive and there will be a painting on the background and your every touch will reflect some effects on the painting.

Input & Output

Touch or click the beat and the beat will be destroyed.

Code

**//Destroy the object itself in a certain time and trigger HP damage if the script is on objects of beats.**

public class natureDestroy : MonoBehaviour  
{  
    *// Use this for initialization*  
    public float lifeTime = 2.0f;  
    public bool healthFlag = false;  
  
    delegate void HPDMG ();  
  
    HPDMG hpdmg;  
  
    void Start ()  
    {  
        Destroy (this.gameObject, lifeTime);  
        if (healthFlag) {  
            hpdmg = GameObject.Find ("HP").GetComponent<hP> ().HpDamage;  
            hpdmg ();  
        }  
    }  
}

**//Control the input interaction and define a class to show “miss” texts while hitting beats at the wrong time or failing to hit beats.**

public class LCtrl : MonoBehaviour  
{  
    public GameObject note;  
    public GameObject circle;  
    public GameObject AfterEffect;  
    public GameObject miss;  
      
    *// Update is called once per frame*  
    void Update ()  
    {  
          
        Collider2D[] col = Physics2D.OverlapPointAll (Camera.main.ScreenToWorldPoint (Input.mousePosition));  
  
        if (col.Length > 0) {  
            foreach (Collider2D c in col) {  
                *//do what you want*  
                if (Input.GetMouseButtonDown (0) && circle.transform.localScale.x < 1.5) {  
                    Destroy (note.gameObject, 0);  
                    Instantiate (AfterEffect, note.transform.position, Quaternion.identity);  
                    GameObject.Find ("HP").GetComponent<hP> ().Healing ();  
                }  
                if (Input.GetMouseButtonDown (0) && circle.transform.localScale.x >= 1.5) {  
                    Instantiate (miss, note.transform.position, Quaternion.identity);  
                    Destroy (note.gameObject, 0);  
                    GameObject.Find ("HP").GetComponent<hP> ().HpDamage ();  
                }   
            }  
        } else {  
            Invoke ("NoteDes", 1.0f);  
        }  
                      
    }  
  
    void NoteDes ()  
    {  
        Destroy (note.gameObject, 0);  
        Instantiate (miss, note.transform.position, Quaternion.identity);  
        GameObject.Find ("HP").GetComponent<hP> ().HpDamage ();  
    }  
}

**//Define classes to damage or heal HP and trigger them at appropriate time.**

public class hP : MonoBehaviour  
{  
    public Slider HPStrip;  
    public int HP = 10;  
    public GameObject panel;  
  
    void Start ()  
    {    
        HPStrip.maxValue = HP;  
        HPStrip.value = HPStrip.maxValue;   
    }  
  
    public void HpDamage ()  
    {    
        HP -= 2;  
        HPStrip.value = HP;  
        if (HP <= 0) {    
            GameObject.Find ("SceneManager").GetComponent<sceneManager> ().GameOver ();  
        }    
    }  
  
    public void Healing ()  
    {  
        if (HP <= 10) {  
            HP += 1;  
            HPStrip.value = HP;  
        }  
    }  
}

**//Generate beats by the rhythm of BGM which is set manually at the voice frequency editor before the game starts.**

namespace SonicBloom.Koreo.Demos  
{  
    [AddComponentMenu ("Koreographer/Demos/Musical Impulse")]  
    public class musicalGtr : MonoBehaviour  
    {  
        [EventID]  
        public string eventID;  
        public GameObject note;  
        public float testScale = 3.5f;  
  
        void Start ()  
        {  
            *// Register for Koreography Events.  This sets up the callback.*  
            Koreographer.Instance.RegisterForEvents (eventID, AddImpulse);  
        }  
  
        void OnDestroy ()  
        {  
            if (Koreographer.Instance != null) {  
                Koreographer.Instance.UnregisterForAllEvents (this);  
            }  
        }  
  
        void AddImpulse (KoreographyEvent evt)  
        {  
            Vector2 r = Random.insideUnitCircle.normalized \* testScale;  
            float ranNum = Random.Range (1.0f, 2.0f);  
            Instantiate (note, new Vector3 (r.x \* ranNum, r.y / ranNum, 0), Quaternion.identity);  
        }  
    }  
}

**//Record the trace of the mouse movements by keeping left mouse down and dragging the cursor to draw a path. The trace will be saved as many points within fixed intervals and the coordinates of points will be saved in List structure.**

[RequireComponent(typeof(MeshCollider))]  
public class ATTraceRecorder : MonoBehaviour {  
    public Camera \_SceneCamera;  
    public int \_MouseBtn = 0;  
    public float \_StepMin = 0.0001f;  
    private Vector3 \_LastPos = new Vector3(float.NegativeInfinity, 0,0);  
    public enum TracePointType  
    {  
        STROKE\_START,  
        STROKE\_POINT,  
        STROKE\_END  
    }  
          
    [System.Serializable]  
    public class TracePoint  
    {  
        public TracePoint(TracePointType t, Vector3 p, TracePoint PrevTP = null)  
        {  
            Type = t;  
            Pos = p;  
            if(PrevTP==null)  
            {  
                Dist = 0.0f;  
            }  
            else  
            {  
                Dist = PrevTP.Dist;  
                Dist += (Pos - PrevTP.Pos).magnitude;  
            }  
        }  
        public TracePointType Type;  
        public Vector3 Pos;  
        public float Dist;  
    }  
    public List<TracePoint> \_Trace = new List<TracePoint>();  
    public List<ATTraceRecorder> \_SKRecs = new List<ATTraceRecorder>();  
  
    *// Use this for initialization*  
    void Start () {  
        if (\_SceneCamera == null) {  
            \_SceneCamera = Camera.main;  
        }  
    }  
      
    *// Update is called once per frame*  
    void Update () {  
          
    }  
  
    void OnMouseDown()  
    {  
        bool bMouse = Input.GetMouseButton (\_MouseBtn);  
        Vector3 hitPos = Vector3.zero;  
        if (HitTestPosition (out hitPos) && bMouse) {  
            AddTracePoint (TracePointType.STROKE\_START, hitPos);  
            *//Debug.Log ("OnMouseDown"+ hitPos);*  
        }  
    }  
  
    void OnMouseDrag()  
    {  
        Vector3 hitPos = Vector3.zero;  
        if (HitTestPosition (out hitPos)) {  
            AddTracePoint (TracePointType.STROKE\_POINT, hitPos);  
            *//Debug.Log ("OnMouseDrag"+ hitPos);*  
        }  
    }  
  
    void OnMouseOver()  
    {  
        bool bMouse = Input.GetMouseButton (\_MouseBtn);  
        Vector3 hitPos = Vector3.zero;  
        if (HitTestPosition (out hitPos) && bMouse) {  
            AddTracePoint (TracePointType.STROKE\_POINT, hitPos);  
            *//Debug.Log ("OnMouseOver"+ hitPos);*  
        }  
    }  
  
    void OnMouseUp()  
    {  
        Vector3 hitPos = Vector3.zero;  
        if (HitTestPosition (out hitPos)) {  
            *//Debug.Log ("OnMouseUp"+ hitPos);*  
            AddTracePoint (TracePointType.STROKE\_END, hitPos);  
        }  
    }  
  
    void OnMouseEnter()  
    {  
        bool bMouse = Input.GetMouseButton (\_MouseBtn);  
        Vector3 hitPos = Vector3.zero;  
        if (HitTestPosition (out hitPos) && bMouse) {  
            *//Debug.Log ("OnMouseEnter"+ hitPos);*  
            AddTracePoint (TracePointType.STROKE\_START, hitPos);  
        }  
    }  
  
    void OnMouseExit()  
    {  
        bool bMouse = Input.GetMouseButton (\_MouseBtn);  
        Vector3 hitPos = Vector3.zero;  
        if (HitTestPosition (out hitPos) && bMouse) {  
            AddTracePoint (TracePointType.STROKE\_END, hitPos);  
            *//Debug.Log ("OnMouseExit"+ hitPos);*  
        }  
    }  
  
    void AddTracePoint (TracePointType tptype, Vector3 curPos)  
    {  
        Vector3 move = curPos - \_LastPos;  
        if (tptype == TracePointType.STROKE\_END ||   
            tptype == TracePointType.STROKE\_START) {  
  
        }  
  
        if (move.magnitude > \_StepMin) {  
            AddTracePointAtPos (tptype, curPos);  
        } else {  
            if (tptype == TracePointType.STROKE\_END || tptype == TracePointType.STROKE\_START) {  
                curPos += \_StepMin \* (Vector3)Random.insideUnitCircle;  
                AddTracePointAtPos (tptype, curPos);  
            }  
        }  
        \_LastPos = curPos;  
    }  
  
    void AddTracePointAtPos (TracePointType tptype, Vector3 curPos)  
    {  
        if (\_Trace.Count > 0) {  
            \_Trace.Add (new TracePoint (tptype, curPos, \_Trace [\_Trace.Count - 1]));  
        }  
        else {  
            \_Trace.Add (new TracePoint (tptype, curPos, null));  
        }  
    }  
  
    bool HitTestPosition(out Vector3 pos){  
          
        pos = new Vector3 (float.NaN, 0, 0);  
  
        RaycastHit hit;  
        Vector3 cursorPos = new Vector3 (Input.mousePosition.x, Input.mousePosition.y, 0.0f);  
        Ray cursorRay=\_SceneCamera.ScreenPointToRay (cursorPos);  
        if (Physics.Raycast(cursorRay,out hit,200)){  
            MeshCollider meshCollider = hit.collider as MeshCollider;  
            if (meshCollider != GetComponent<MeshCollider> ())  
                return false;  
            if (meshCollider == null || meshCollider.sharedMesh == null)  
                return false;      
              
            pos = hit.point;  
  
            return true;  
        }  
        else{      
            return false;  
        }  
    }  
  
    public void Save(int Slot)  
    {  
        if (Slot >= \_SKRecs.Count && Slot<0) {  
            *//Debug.Log ("Exceed Slot Count");*  
            return;  
        }  
        \_SKRecs [Slot].\_Trace = this.\_Trace;  
    }  
  
    public void Load(int Slot)  
    {  
        if (Slot >= \_SKRecs.Count && Slot<0) {  
            *//Debug.Log ("Exceed Slot Count");*  
            return;  
        }  
        \_Trace = \_SKRecs [Slot].\_Trace;  
    }  
  
    public void ClearTrace()  
    {  
        \_Trace.Clear ();  
    }  
  
    public bool GetInterpPosAtId(int id, float dist, out Vector3 pos)  
    {  
        pos = new Vector3 (float.NegativeInfinity, 0, 0);  
        if (id >= \_Trace.Count - 1 || id<0) {  
            return false;  
        }  
        *//Debug.Log ("id:" + id);*  
        if(\_Trace [id].Dist > dist)  
        {  
            return false;  
        }  
        if(\_Trace[id+1].Dist <dist)  
        {  
            return false;  
        }  
        Vector3 p0 = \_Trace [id].Pos;  
        Vector3 p1 = \_Trace [id + 1].Pos;  
        float t = (dist - \_Trace [id].Dist) / (\_Trace [id + 1].Dist - \_Trace [id].Dist);  
        pos = Vector3.Lerp(p0,p1,t);  
  
        return true;  
    }  
  
    public bool GetNextIdBeforeDist(float dist, int fromId, out int nextId)  
    {  
        nextId = -1;  
        if (fromId >= \_Trace.Count - 1) {  
            *//Debug.Log ("fromId >= \_Trace.Count - 1");*  
            return false;  
        }  
  
        if (\_Trace [fromId].Dist > dist) {  
            *//Debug.Log ("\_Trace [fromId].Dist > dist");*  
            bool bJust = IsDistJustAfterId (dist,fromId);  
            return false;  
        }  
  
        int nxt = fromId;  
        while (\_Trace [nxt].Dist < dist) {  
            nxt += 1;  
            *//Debug.Log ("nxt:" + nxt);*  
            if (nxt >= \_Trace.Count - 1) {  
                *//Debug.Log ("Exceed!");*  
                return false;  
            }  
        }  
        nextId = nxt-1;  
        return true;  
          
    }  
  
    public bool IsDistJustAfterId(float dist, int id)  
    {  
        if (id < 0 || id > \_Trace.Count - 2) {  
            return false;  
        }  
  
        if (\_Trace [id].Dist <= dist && \_Trace [id + 1].Dist >= dist) {  
            return true;  
        } else {  
            return false;  
        }  
  
    }  
  
    public bool DistExceedTrace(float dist)  
    {  
        if (\_Trace.Count < 2) {  
            return false;  
        }  
  
        return \_Trace [\_Trace.Count - 1].Dist < dist;  
    }  
}

**//trace the path of the recorder**

public class trace : MonoBehaviour  
{  
  
    public ATTraceRecorder \_Recorder;  
    int id = 0;  
    *// Use this for initialization*  
    void Start ()  
    {  
  
    }  
  
    *// Update is called once per frame*  
    void Update ()  
    {  
        if (id < \_Recorder.\_Trace.Count) {  
            transform.position = \_Recorder.\_Trace [id].Pos;  
            id++;  
        }  
  
    }  
}

**//The effect of brush on background with mouse interaction**

public class 裂星模式 : MonoBehaviour  
{  
  
    *// Use this for initialization*  
    public Vector3 offset;  
    public float \_ForcePosDist = 1.0f;  
    public float \_ForceScale = 0.5f;  
    Vector2 \_Force;  
    Rigidbody2D rb;  
  
    void Start ()  
    {  
        rb = GetComponent<Rigidbody2D> ();  
        \_Force = Random.insideUnitCircle.normalized;  
        rb.velocity = \_Force \* \_ForceScale;  
    }  
}

Illustration