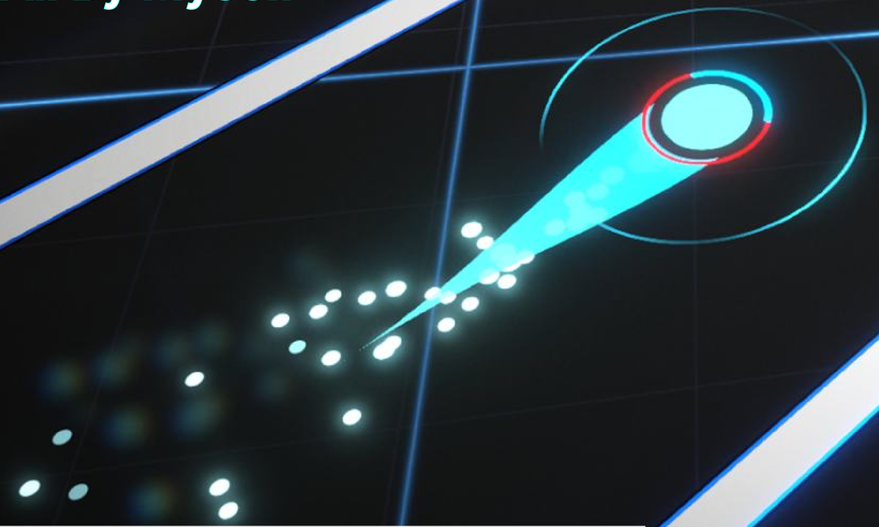
Made With

Made All By Myself

https://flandrescarlet-15532.itch.io/meta-clash

Demo Link



Meta Clash is a roguelike action game with unique mechanics based on physics.
Players control a ball to attack their enemies.

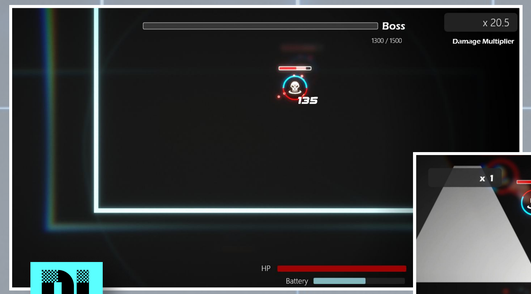
Meta Clash

Reference

The gameplay was inspired by the Windows game “3D Pinball”. Snce there was a lack of games with new mechanics on the market, I decided to combine the core gameplay of “3D Pinball” with action game elements.

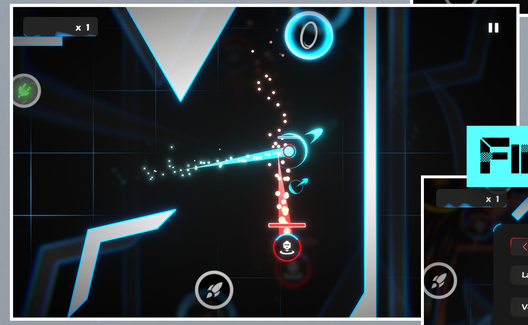
The game's art style is inspired by the movie “Tron: Legacy”, and is set in a futuristic metaverse.

Iteration

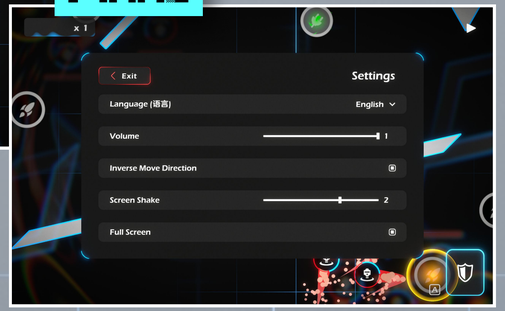


01

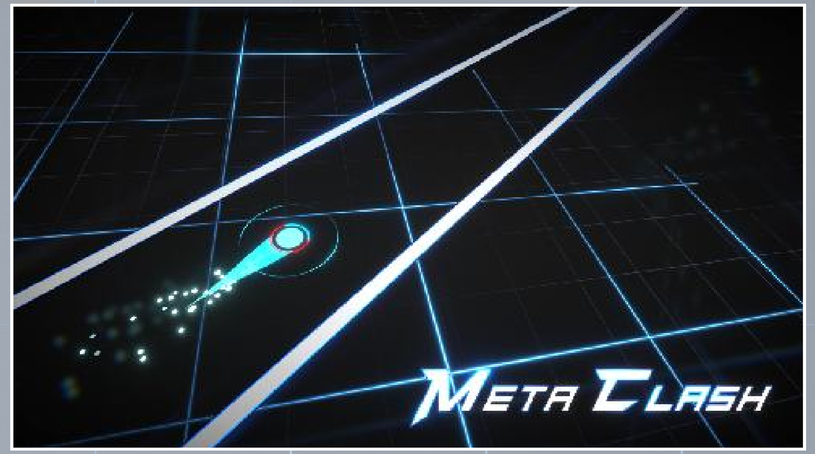
02

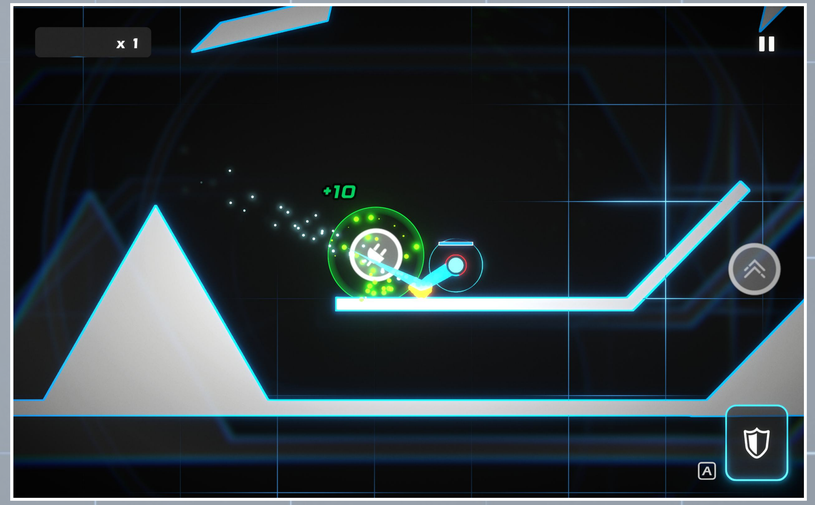


FINAL



Screenshots





Gameplay

Move

Drag your mouse/finger at any position (except for places occupied by buttons) to control the direction of your ball.

Attack

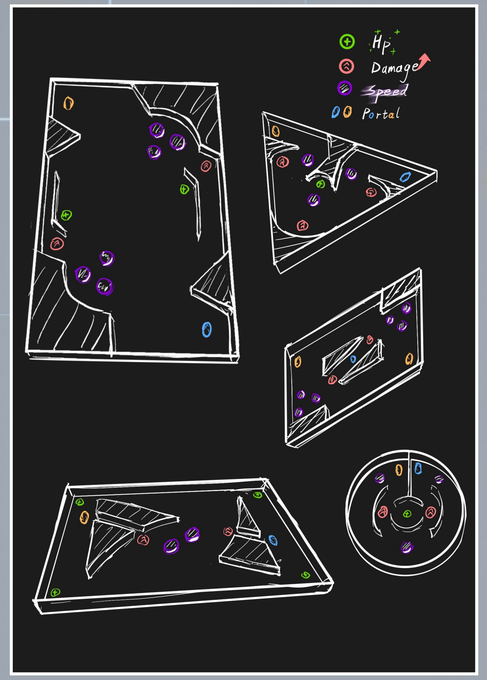
When the blue part of your ring collides with the red part of a enemy, you deals damage to it based on your Damage Multiplier and relative speed.

Parry

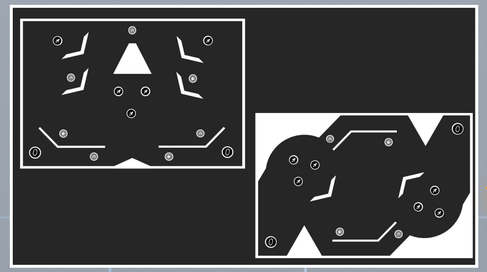
When a enemy is about to hit the player, an alert will appear. The player can use the parry skill by pressing corresponding buttons. Parry deals more damage than normal attack.

Damage Multiplier Shown in the top-left corner. Directly affects the damage players cause.

Map Design

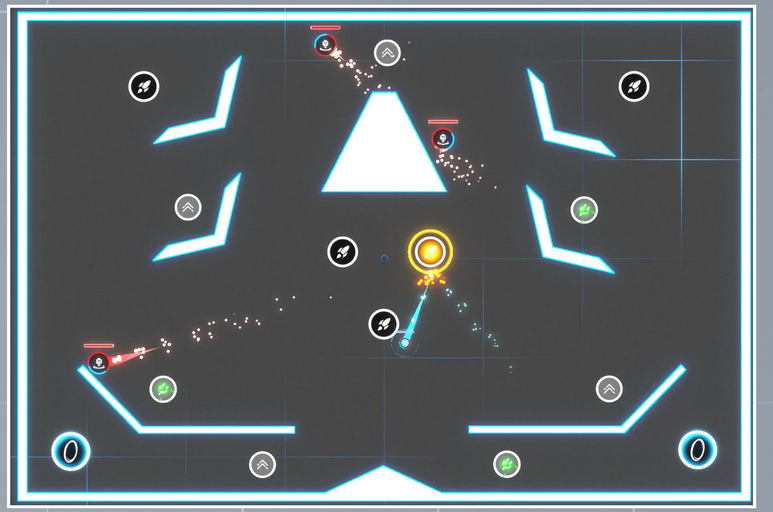


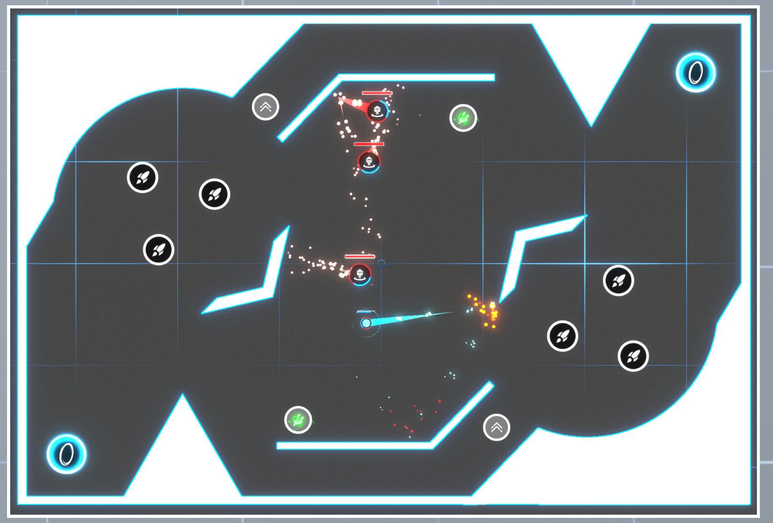
Sketch



Final Version

In-Game Overview



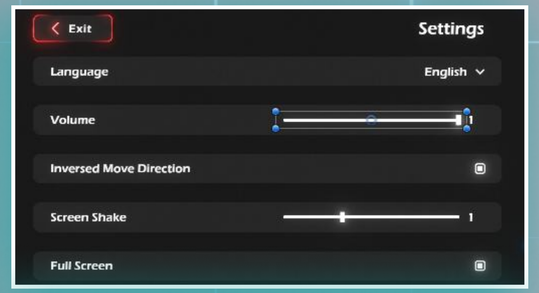


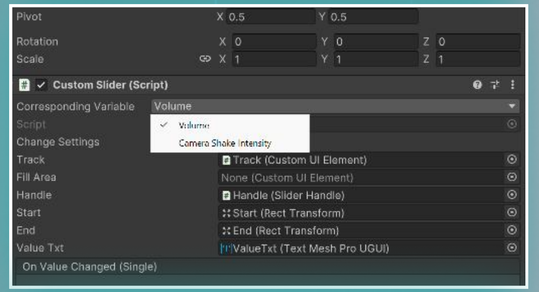
Code

-Settings System

Using Reflection to get all the variables in the settings that match the component, and add a custom Popup to set the corresponding settings item of an UI with just two clicks.

```
[CustomEditor(typeof(CustomSlider))]
public class SliderInspector : Editor{
    public override void OnInspectorGUI(){
        CustomSlider slider = (CustomSlider)target;
        slider.UpdateUI();

        if (slider.changeSettings){
            EditorGUILayout.BeginHorizontal();

            EditorGUILayout.LabelField("Corresponding Variable", GUILayout.Width(150));

            var allFields = typeof(SettingsData).GetFields();
            List<string> fields = new();
            foreach (var f in allFields){
                if (f.FieldType == typeof(float) || f.FieldType == typeof(int) || f.FieldType == typeof(double)){
                    fields.Add(f.Name);
                }
            }

            var names = new string[fields.Count];
            int i = 0;
            foreach (var field in fields){
                names[i] = Tools.ToNormalEnglish(field);
                i++;
            }

            int selected = fields.IndexOf(slider.variableName);
            if (selected == -1){
                selected = 0;
            }

            selected = EditorGUILayout.Popup(selected, names);
            slider.variableName = fields[selected];

            EditorGUILayout.EndHorizontal();
        }
        else{
            slider.variable = null;
        }

        base.OnInspectorGUI();
    }
}
```





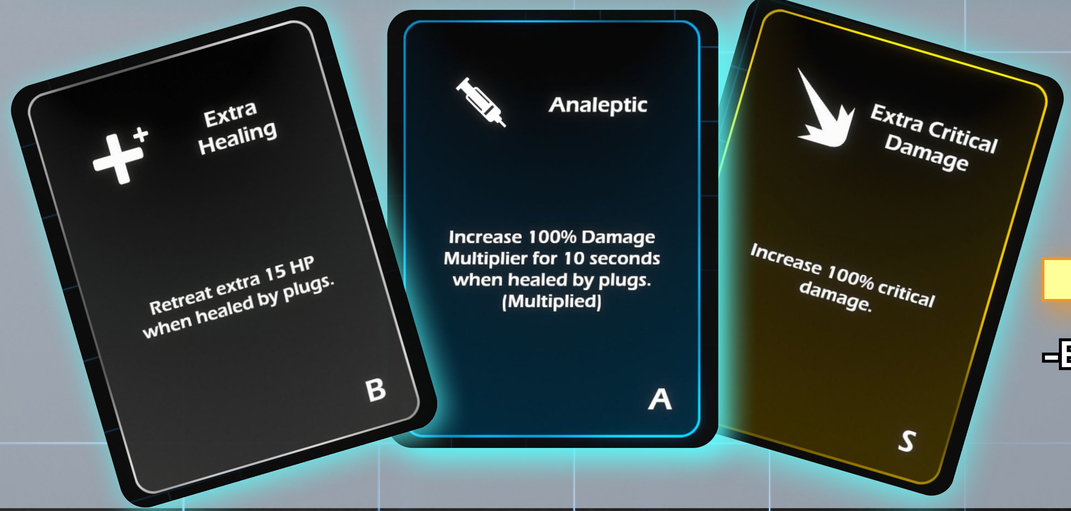

```
public float value{
    get{ return _value; }
    set{
        float tmp = Mathf.Clamp(value, (float)Math.Round(value, roundAccuracy), minValue, maxValue);
        if (tmp != _value){
            SoundSys.PlaySound(name: "SFX/Slider");
            _value = tmp;
            if (onValueChanged != null){
                onValueChanged.Invoke(_value);
            }

            if (variable != null){
                variable.SetValue(obj: Settings.data, _value);
                Settings.SaveSettings();
            }

            UpdateUI();
        }
    }
}
```

```
public class SettingsData : ICloneable{
    public bool inverseMouseDirection;
    public bool fullScreen;
    public float volume;
    public float cameraShakeIntensity;
    public Language language;

    public void Init(){
        inverseMouseDirection = false;
        cameraShakeIntensity = 1.5f;
        language = Application.systemLanguage == SystemLanguage.Chinese ? Language.Chinese : Language.English;
        fullScreen = false;
        volume = 1.0f;
    }

    public object Clone(){
        return base.MemberwiseClone();
    }
}
```

```
public class BuffSys{
    public Dictionary<string, List<Buff>> stringPairs;
    public Dictionary<BuffTriggerEvent, UnityEvent<object[]>> events;
    public Unit unit;

    public BuffSys(Unit u){
        unit = u;
        stringPairs = new Dictionary<string,List<Buff>>();
        events = new Dictionary<BuffTriggerEvent,UnityEvent<object[]>>();
    }

    public void AddBuff(Buff b){
        b.unit = unit;
        if (stringPairs.ContainsKey(b.name)){
            if (b.cumulateMode == BuffCumulateMode.AddLayer){
                if (b.layerLimit == 0 || b.layerLimit > stringPairs[b.name].Count){
                    stringPairs[b.name].Add(b);
                    //allBuffs[b.triggerEvent].Add(b);
                }
            }
            else if (b.cumulateMode == BuffCumulateMode.Refresh){
                if (b.lastingType == BuffLastingType.Lasting){
                    if (b.duration > stringPairs[b.name][0].durationLeft){
                        stringPairs[b.name][0].durationLeft = b.duration;
                    }
                }
                else if (b.lastingType == BuffLastingType.Triggered){
                    if (b.maxTriggerCount > stringPairs[b.name][0].maxTriggerCount){
                        stringPairs[b.name][0].maxTriggerCount = b.maxTriggerCount;
                    }
                }
            }
        }
        else{
            stringPairs.Add(b.name, new List<Buff>(){ b });
        }

        if (b.triggerEvent == BuffTriggerEvent.Immediate){
            b.Trigger();
        }
    }
```

Using two Dictionaries to store Buffs. Easy to manage and expandible.

```
if (name == "Speed Up"){
    int value = 2;
    if (rank == 'A'){
        value = 3;
    }
    else if (rank == 'S'){
        value = 5;
    }

    return new Buff(out b, name, displayedName: LocalizeTxt(params texts: "Speed Up", "加速"), BuffCumulateMode.AddLayer, BuffLastingType.Permanent, BuffTriggerEvent.MatchStarted,
        actionTriggered: () => {
            var u = b.unit.entity;
            u.rb.linearVelocity += u.direction * value;
        },
        descript: LocalizeTxt(params texts: $"Increase the initial speed by {value} m/s.",
            $"提高 {value} m/s 初始速度")
    );
}
if (name == "Extra Damage"){
    int value = 25;
    if (rank == 'A'){
        value = 50;
    }
    else if (rank == 'S'){
        value = 100;
    }

    return new Buff(out b, name, displayedName: LocalizeTxt(params texts: "Extra Damage", "伤害加成"), BuffCumulateMode.AddLayer, BuffLastingType.Permanent, BuffTriggerEvent.Immediate,
        actionTriggered: () => { b.unit.percentBuff[UnitAttribute.DamageMult] += value / 100f; },
        actionDeleted: () => { b.unit.percentBuff[UnitAttribute.DamageMult] -= value / 100f; },
        descript: LocalizeTxt(params texts: $"Increase {value}% damage multiplier (Multiplied).",
            $"提高 {value}% 伤害倍率（乘算）")
    );
```

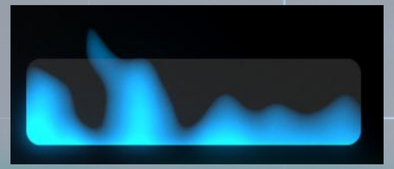


Code

-Buff System

Technical Art

-2D Fire



```
struct v2f{
    float2 uv : TEXCOORD0;
    float4 vertex : SV_POSITION;
    float4 color : COLOR;
};

sampler2D _MainTex;
float4 _MainTex_ST;
float _Emission, _Edge;
float4 _Color;
float2 _Scale, _Speed;

v2f vert(appdata v){
    v2f o;
    o.vertex = UnityObjectToClipPos(v.vertex);
    o.uv = TRANSFORM_TEX(v.uv, _MainTex);
    o.color = v.color;
    return o;
}

float4 frag(v2f i) : SV_Target{
    float h = (1 - pow(i.uv.y, 0.5)) - (1.5 - _Edge);
    float2 uv = (i.uv + _Time * _Speed * -1) * (1 / _Scale);
    float fire = GradientNoise(uv, Scale 10) * SimpleNoise(uv, Scale 10);

    float a = tex2D(_MainTex, i.uv).a * clamp(fire + h, 0, 1);
    float4 color = _Color * (_Emission + 1);
    return float4(color.rgb, a) * i.color;
}
ENDCG
```

I found that Shader Graphs don't support _Stencil property, so I wrote a shader myself instead of using shader graphs. The Simple noise and Gradient noise are methods from Unity's shader graph document.

Simple Noise Node | Shader Graph | 10.5.1 (unity3d.com)

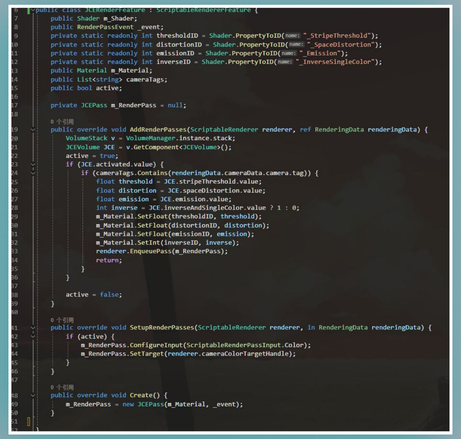
Gradient Noise Node | Shader Graph | 10.5.1 (unity3d.com)

Technical Art

Video: https://youtu.be/7urWfuJ2_mM

-Custom Full Screen Post-Processing & Timeline

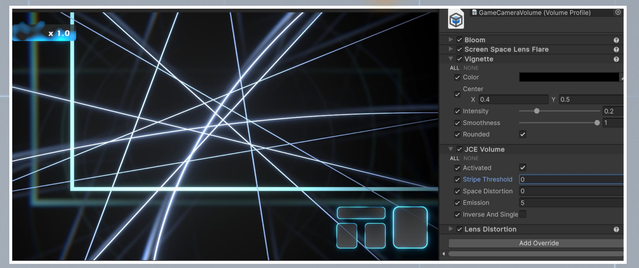
I used a custom Render Feature, a custom Render Pass to make a Volume, and used a Shader to implement a custom full screen effect. I then used Unity's Timeline system to implement the "Judgement Cut End" effect using Unity's Timeline.



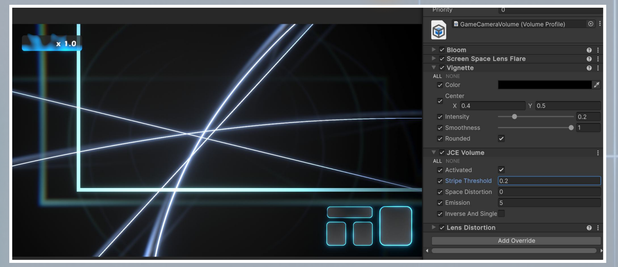
```
public class JCERenderFeature : ScriptableRendererFeature {
    public Shader m_Shader;
    public RenderPassEvent _event;
    private static readonly int thresholdID = Shader.PropertyToID("_StripeThreshold");
    private static readonly int distortionID = Shader.PropertyToID("_SpaceDistortion");
    private static readonly int emissionID = Shader.PropertyToID("_Emission");
    private static readonly int inverseID = Shader.PropertyToID("_InverseSingleColor");
    public Material m_Material;
    public List<string> cameraTags;
    public bool active;

    private JCEPass m_RenderPass = null;

    0 个引用
    public override void AddRenderPasses(ScriptableRenderer renderer, ref RenderingData renderingData) {
        VolumeStack v = VolumeManager.Instance.stack;
        JCEVolume JCE = v.GetComponent<JCEVolume>();
        active = true;
        if (JCE.activated.value) {
            if (cameraTags.Contains(renderingData.cameraData.camera.tag)) {
                float threshold = JCE.stripeThreshold.value;
                float distortion = JCE.spaceDistortion.value;
                float emission = JCE.emission.value;
                int inverse = JCE.inverseAndSingleColor.value ? 1 : 0;
                m_Material.SetFloat(thresholdID, threshold);
                m_Material.SetFloat(distortionID, distortion);
                m_Material.SetFloat(emissionID, emission);
                m_Material.SetInt(inverseID, inverse);
                renderer.EnqueuePass(m_RenderPass);
                return;
            }
        }

        active = false;
    }

    0 个引用
    public override void SetupRenderPasses(ScriptableRenderer renderer, in RenderingData renderingData) {
        if (active) {
            m_RenderPass.ConfigureInput(ScriptableRenderPassInput.Color);
            m_RenderPass.SetTarget(renderer.cameraColorTargetHandle);
        }
    }

    0 个引用
    public override void Create() {
        m_RenderPass = new JCEPass(m_Material, _event);
    }
}
```

Custom Render Feature

Stripe Threshold = 0

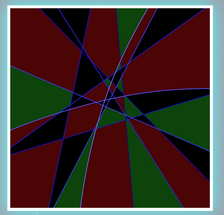


Stripe Threshold = 0.2

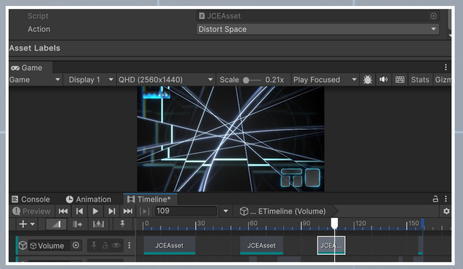


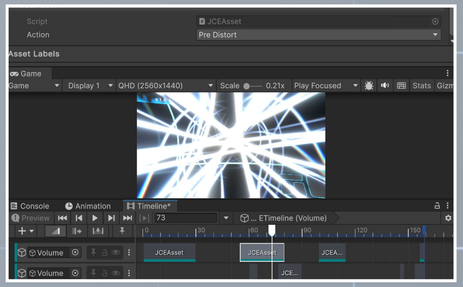
Space Distortion = 5

Main Texture



Timeline



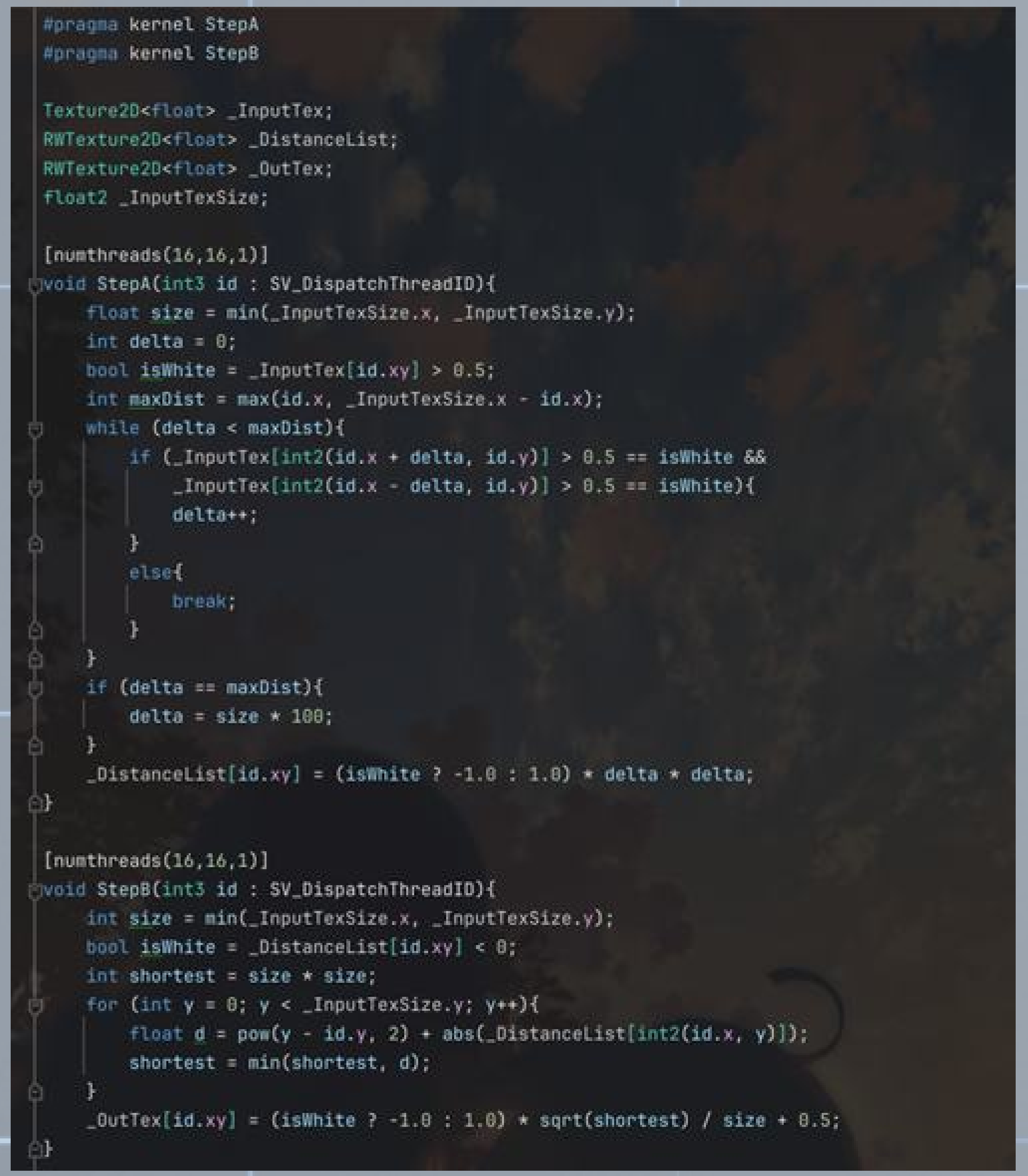


Technical Art

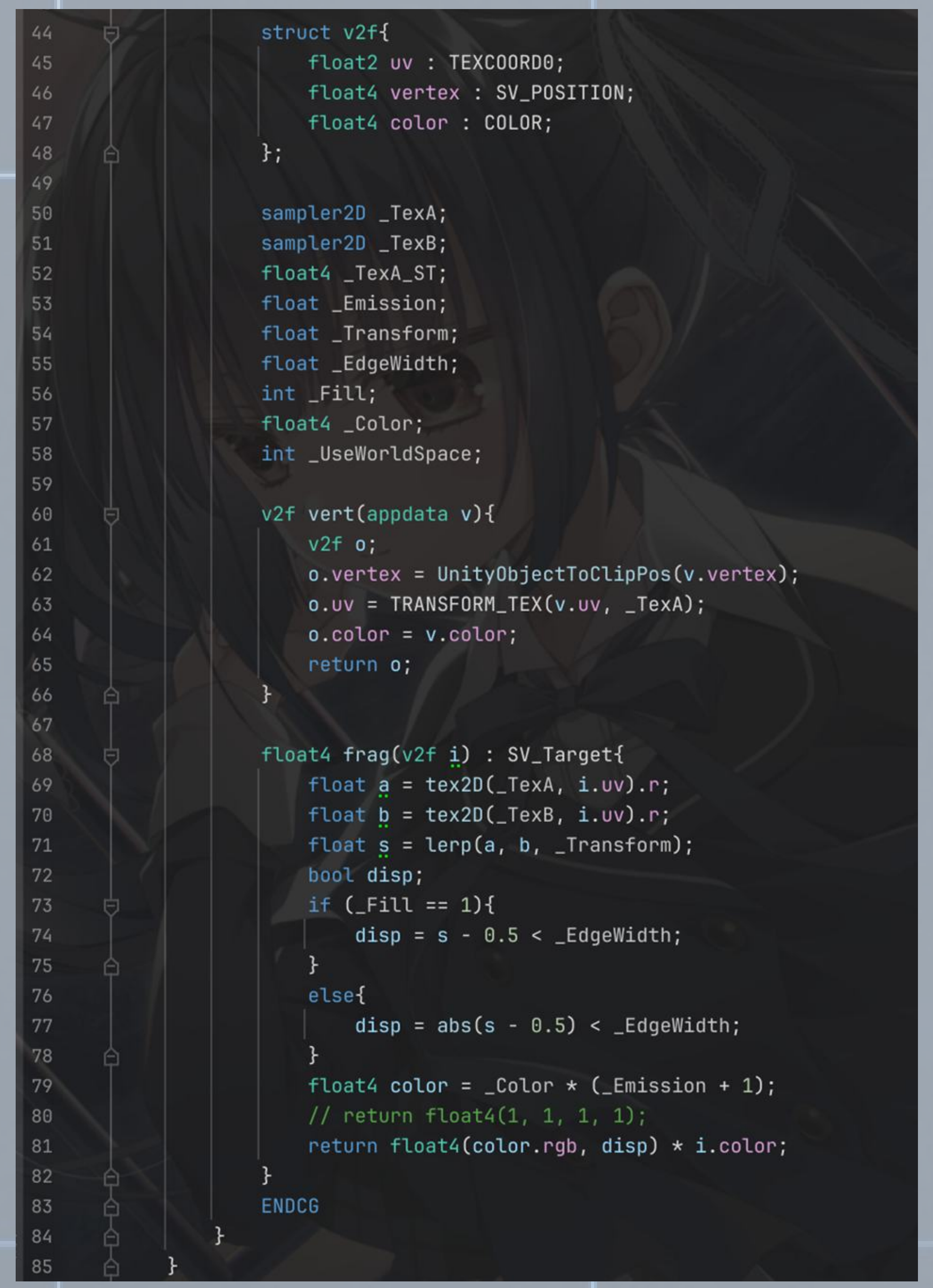
- 2D SDF Texture Generation with Compute Shaders

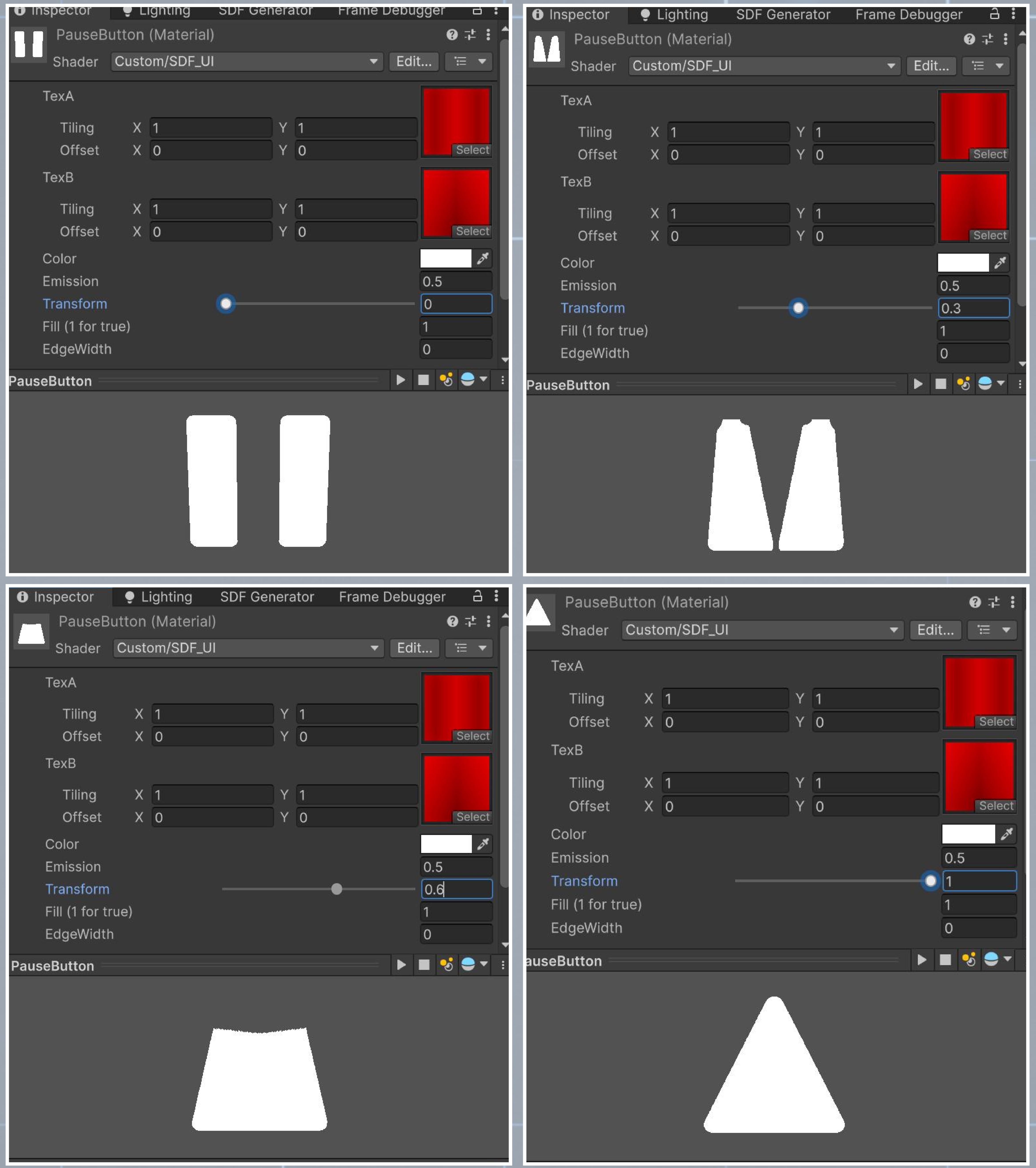
I wrote a Compute Shader using the “Two Pass” method to generate Signed Distance Field texture for icons. Then, I wrote a Shader to make the transition animation of the Pause/Continue icon.

Compute Shader



Shader for Icon





Summary

In this project, I got a deeper understanding of shaders and VFX Graphs and learned how to use features such as Reflection, Customized Inspector, Timeline, Compute Shaders, Custom Render Feature and so on. I've also learned how to support mobile phones and touchscreen.

Based on some playtests, I'm planning to add more maps and buffs to increase variability. Besides, I'm planning to add save/load systems to allow players continue a game even if they have to leave in the middle.