

学 号	0122005830214
-----	---------------

武汉理工大学

计算机图形学大作业

题 目	基于 Unity 的多平台 2D 宇宙模拟系统
学 院	理学院
专 业	信息与计算科学
班 级	信计 2001 班
姓 名	石一佐
指导教师	陆济湘
成 绩	

2023 年 6 月 25 日

一、摘要

当下，Unity 作为一个优秀的实时互动内容创作和运营平台，正在越来越多地被各个层次的游戏开发者所使用，例如国内知名游戏《王者荣耀》、《原神》、《炉石传说》等皆由 Unity 引擎开发。本文将介绍个人从零开发的一个简易的宇宙模拟系统。从底层来说实现了星球的生成、星球之间的引力、碰撞等要素，游戏玩法上实现了随机放置、按条件放置、删除、初始化等操作，界面优化上实现了星球轨道的绘制、UI 的制作、背景音乐的实现。鉴于 Unity 的多平台开发，通过 Unity 多平台操作绑定，本游戏在 Android 平台上也能正常运行。

二、项目介绍

本项目可以从我的 GitHub 上获取：<https://github.com/164K/Universe>，目前包括 Windows 与 Android 的发行版本、源代码以及基本玩法。

2.1 需求分析

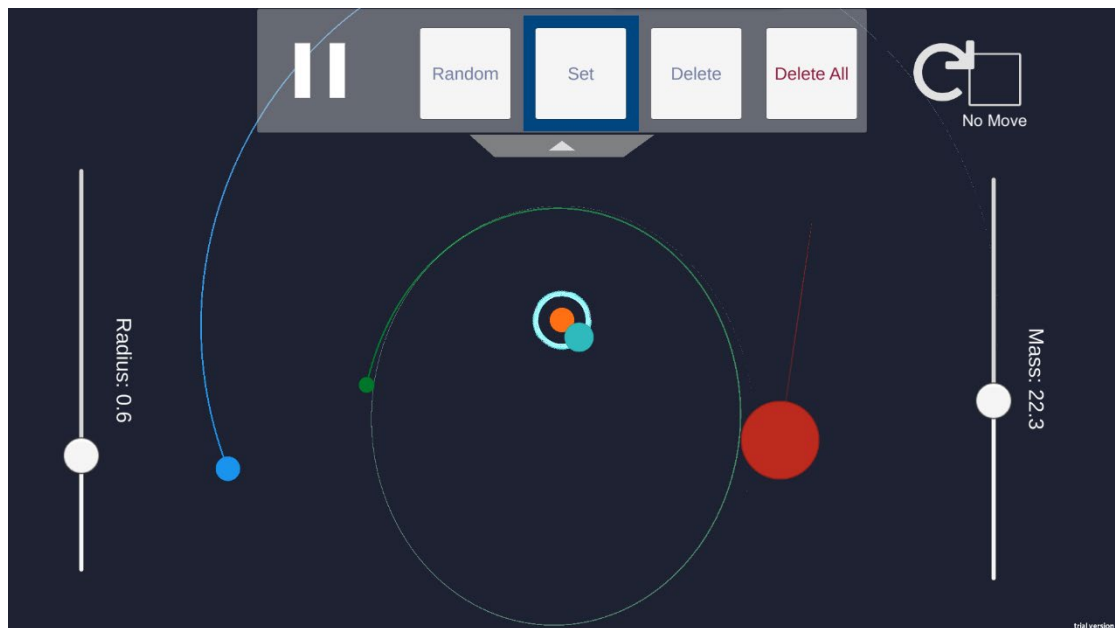
如今游戏层出不穷，但是当下纷繁复杂的游戏太多，纯粹、无目的的休闲类太少。实现一个简单、放松类的游戏，工作学习空闲之余玩一玩，这样的需求很适合工作或者比较忙碌的人群。特别是最近轻量化、简洁化的游戏、软件越来越受到开发者与玩家们的青睐，这款游戏便是基于此需求而创作出来的。


另外，随着《三体》、《流浪地球》等科幻文学影视作品的推广，星球、宇宙要素越来越受消费者的喜爱。

2.2 基本玩法

Random: 初始模式（**Random**）下，会默认创建一个“地球”与“太阳”。按住创建一个随机半径、质量的星球，按住同时拖拽可以获得初始速度，可以选择方向，拖拽越大初速度越大。

选择面板: 上方面板点击下拉面板选择功能，初始为 Random 随机模式，**Set** 模式下可以选择星球的质量、半径、是否移动等选项：



Delete: Delete 模式下点击已经存在的星球会删除它；按下刷新按钮会重新加载场景到初始模式下，包括背景音乐。

Delete All: 删除所有已经存在的星球。

暂停: 按下暂停按钮会暂停所有星球的运动，但会保存其初始状态；暂停时按下开始按钮恢复。暂停时可以放置星球并暂时存储初速度，恢复时释放速度。

2.3 功能模块

实现的功能与技术栈如下。基本按照从底层到上层的顺序介绍：

1. 2D 精灵设置 使用 Unity 的 2D Sprite 制作星球的预制件，方便后续放置；

2. 引力系统 即星球之间相互引力;
3. 输入系统 实现各平台点击输入与绑定;
4. 事件系统 输入系统的基础上绑定事件监听;
5. 功能实现 实现基本的放置、删除、重新加载等功能;
6. 界面优化 实现星球轨迹的绘制、拖拽直线的绘制;
7. 制作 UI 制作相应的 UI 绑定基本功能;
8. 动画制作 实现面板的上下运动;
9. 音效添加 添加背景音乐;
10. 多平台视觉优化 针对于多平台适当调整锚点、对齐使得多平台视觉一致;
11. 游戏状态管理 从玩家的角度, 通过整个游戏的生命周期, 实现游戏的状态管理, 对不同的状态实现不同的功能、UI 交互等。

三、开发内容

3.1 星球预制件的创建与引力系统

3.2 基本组件

Sprite Renderer 渲染星球 2D 图像, 星球采用的是 Circle 圆;

Circle Collider 2D 使星球之间可以发生碰撞;

Rigid Body 2D 拥有质量、可以被力所作用。

Line Renderer 线渲染器, 用于拖拽生成时显示拖拽路径;

Trail Renderer 轨迹渲染器, 用于渲染星球的轨迹;

Plant 星球作为该游戏唯一实例化实体, 其预定义属性十分重要。为星球单独创

建一个类 Planet 组件，其中包含如下基本属性（注：部分属性通过 C#的属性 get 与 set 方法来设置，而不是单独存储一个值）：

属性名称	类型	属性功能	备注
Position	Vector2	星球位置	可以从 Transform 获得
Velocity	Vector2	星球速度	可以从 Rigid Body 2D 中获得
PlanetColor	UnityEngine.Color	星球颜色	
TrailColor	UnityEngine.Color	轨迹颜色	通过设置 TrailRenderer
IsKinematic	Bool	是否设置为动力学（不会移动）	可以从 Rigid Body 2D 中设置
IsNoMove	Bool	是否不移动	为一个特定状态，将循环设置 IsKinematic
Mass	Float	质量	可以从 Rigid Body 2D 中获得
Radius	Float	半径	可以从 transform.localScale 设置、获取

Pauser 暂停器，用于拖拽生成时暂停所有星球的运动，并将速度值存储在组件中，开始时读取数值运动。

3.3 实例化与特质化

通过预制件，可以使用 Instantiate 函数在指定 GameObject 下生成一个子 GameObject，然后通过 Plant 脚本中自建的 Initialized 函数设置其位置、速度等基本信息。这样就可以实例化一个星球对象。

可以通过该预制件修改一些初始参数从而得到一些自定义特殊的星球，例如地球、太阳的设定，以及黑洞（设定碰撞删除）。

3.4 引力系统

引力系统是星球交互运动关键，Rigid Body 自带的重力系统只能生成垂直向下的重力场。为了实现星球之间的引力，需要建立一个组件专门管理受引力影响的星球集合。在每一个物理帧（FixUpdate 而不是 Update）中对每个星球施加引力。这会两层遍历所有的星球，时间复杂度为 $O(n^2)$ 。由万有引力定律可知，两个星球 $P_1(x_1, y_1)$ 与 $P_2(x_2, y_2)$ 之间的引力大小为：

$$F = \frac{Gm_1m_2}{r^2}$$

m_1, m_2 为 P_1, P_2 的质量，G 为引力常量，本游戏中取 1。其中 r 为两个星球的欧式距离，即

$$r = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

力的方向指向两个物体连线的内侧，即

$$\begin{aligned} \mathbf{F}_1 &= \frac{Gm_1m_2\mathbf{r}_{1 \rightarrow 2}}{|\mathbf{r}_{1 \rightarrow 2}|^3} \\ \mathbf{F}_2 &= \frac{Gm_1m_2\mathbf{r}_{2 \rightarrow 1}}{|\mathbf{r}_{2 \rightarrow 1}|^3} = -\mathbf{F}_1 \end{aligned}$$

其中 $\mathbf{r}_{a \rightarrow b} = (x_b - x_a, y_b - y_a)$ 。

游戏测试中发现可能会出现两星球距离十分小的情况，这时引力趋近于无穷大，这会导致报错。于是为了加以限制，会设定判定距离的最小值 EPS，小于 EPS 将视作 EPS，即

$$r^* = \max\{r, EPS\}$$

本游戏取 $EPS=10^{-5}$ 。

定义了引力就可以创建一个初始的圆周运动场景：地球围绕太阳作圆周运动。

根据万有引力定律以及向心力公式，圆周运动的速度大小为：

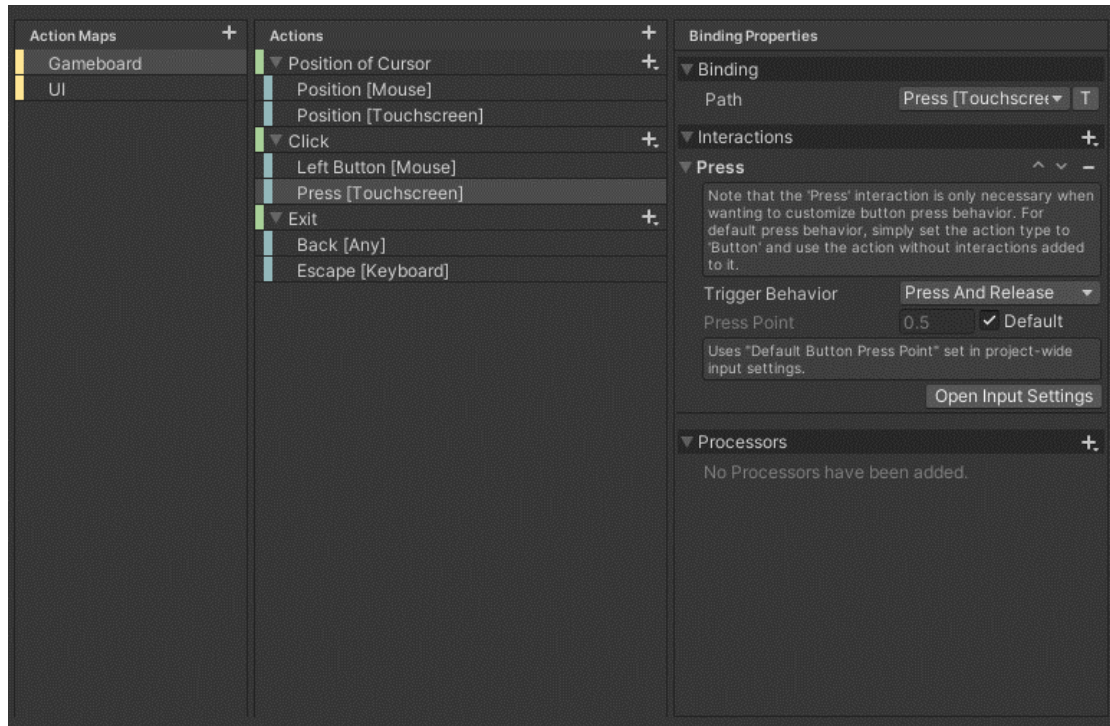
$$v = \sqrt{\frac{GM}{r}}$$

这样就可以构建一个最简单的行星系统了。

3.2 输入系统的处理与事件监听

3.2.1 Input System

一般的 UI 系统会自带有相应的输入监测系统，它能完成一些比较简单的点击事件。但是本游戏需要实现一些比较复杂的输入系统，比如获取点击的位置，鼠标按下、抬起分别检测等，这些需要对输入系统进行细化；另外为了解决跨平台输入检测的问题，例如 Android 的触控点击与 Windows 鼠标点击，都属于同一属性，分散开发过于麻烦。而 Unity 的 Input System 就能解决上述问题，它拥有所有平台、几乎所有按键的基本设定，专门有组件来绑定特定的操作（包中称为“行动”Action）而不需要考虑平台；还能细粒化操作比如按钮按下、抬起等。本游戏使用的 Action 的用法有三种，第一是硬件位置获取，例如获取鼠标指针所在位置；一种是状态检测，例如按下阶段还是抬起（isPressed），在每个 Update 中检测状态不断更新；最后一种是事件监听，即在按下瞬间或者抬起瞬间执行函数，函数需要在一开始注册。



上方图片为 Input Actions，是管理 Action 的组件。左侧是采用的所有 Action Maps，即所用的不同场景。中间是该 Map 的所有 Actions，每个 Action 可以绑定多个平台的操作。例如 Click 点击行为，绑定了鼠标的左键，也绑定了触摸屏的按下。右侧具体化相应的交互行为。上述定义下 Click.started 为刚按下时事件监听器，Click.performed 为按下后抬起事件监听器。PositionOfCursor 可以获得当前鼠标或触摸的位置。

3.2.2 UI 与游戏面板区域与物体检测

虽然此时可以实现点击面板位置的提取，但是点击的位置在哪里也需要考虑。

不然就可能使得点击 UI 却在 UI 下面图层生成一个星球。

使用 Physics2D.Raycast 可以获得物理光线投射，通过鼠标位置获得投射的物体。

获取之后便可以分辨 UI 与游戏面板，并且还可以获得点击的星球以选择、删除。

上面所说的都是针对最普遍的事件检测，至于 UI，UI 有很多内置的事件监听系

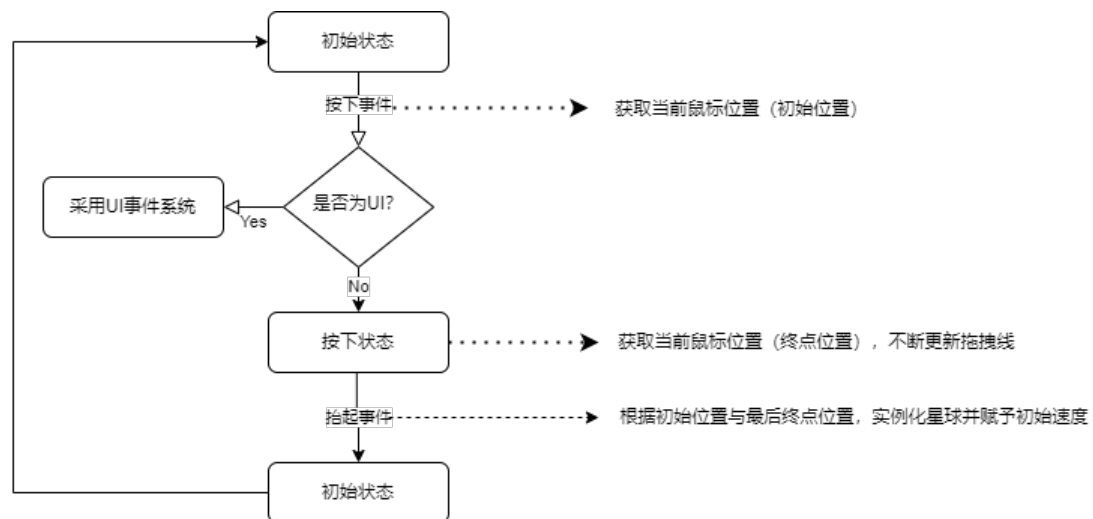
统，并不需要额外定义事件。

3.3 基本游戏功能的实现

拥有了前面的若干基础系统，基本功能的实现便容易得到。

3.3.1 点击创建

点击创建星球由于需要拖拽、点击、抬起多个事件的细粒度，相对比较复杂，涉及一些状态变化。基本流程图如下：



3.3.2 点击删除

点击删除的事件操作相对比较简单，只需要用点击事件检测就可以了；物体检测可以用上面提到的游戏物体检测。这样就能确定删除一个物体。

另外需要注意的是，本游戏删除物体使用的是 `SetActive(false)`，并不是直接删除这个 `GameObject`，因为部分事件监听系统是独立的，删除后无法绑定物体，可能会产生一些错误。

3.3.3 全屏删除

这个很容易实现，只需要将所有星球的父物体中迭代删除所有 Planet 即可。

注意 Delete All 与 Delete 实现有所不同，这里才是删除所有物体，因为此时点击在 UI 上，不会产生不确定的错误，而且可以释放空间。

3.3.4 重新导入

重新导入场景即可。使用的是

```
SceneManager.LoadScene(SceneManager.GetActiveScene().name);
```

语句。

3.3.5 暂停与开始

使用前面所述的 Pauser 即可完成该功能，暂停时调用所有 Planet 下 Pauser 来 Pause 并存储速度等参数。

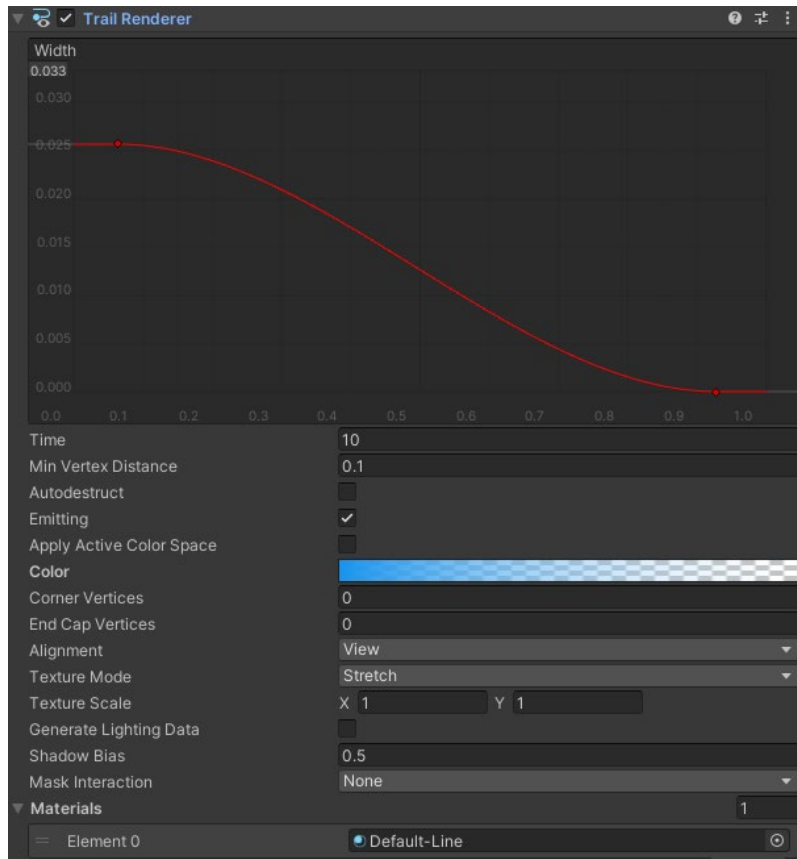
3.3.6 退出游戏

Exit Action 后即可退出，与 Exit 绑定的操作有 Windows 的 Esc 键，Android 的返回导航键。

3.4 界面优化

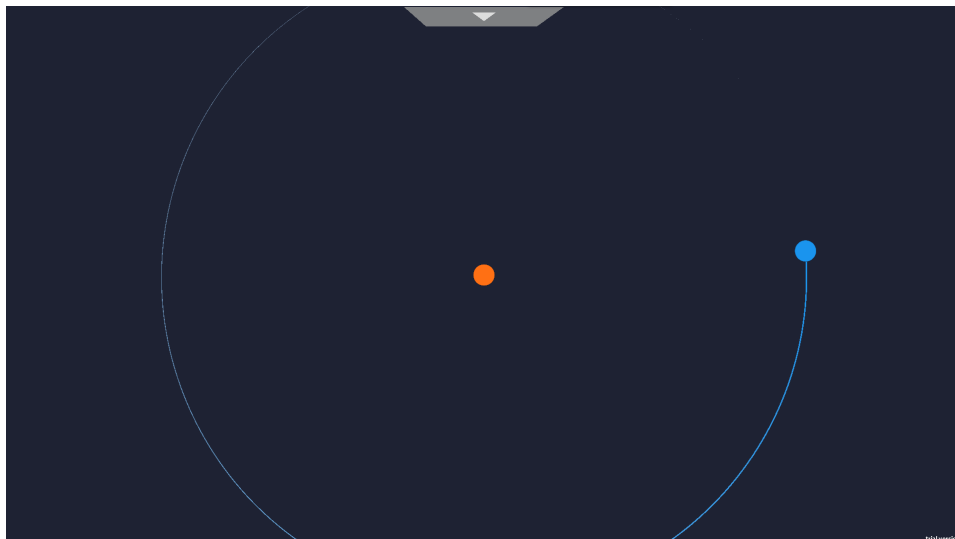
3.4.1 轨迹绘制

Unity 有专用的轨迹渲染器，可以设置线粗细变化、颜色渐变。



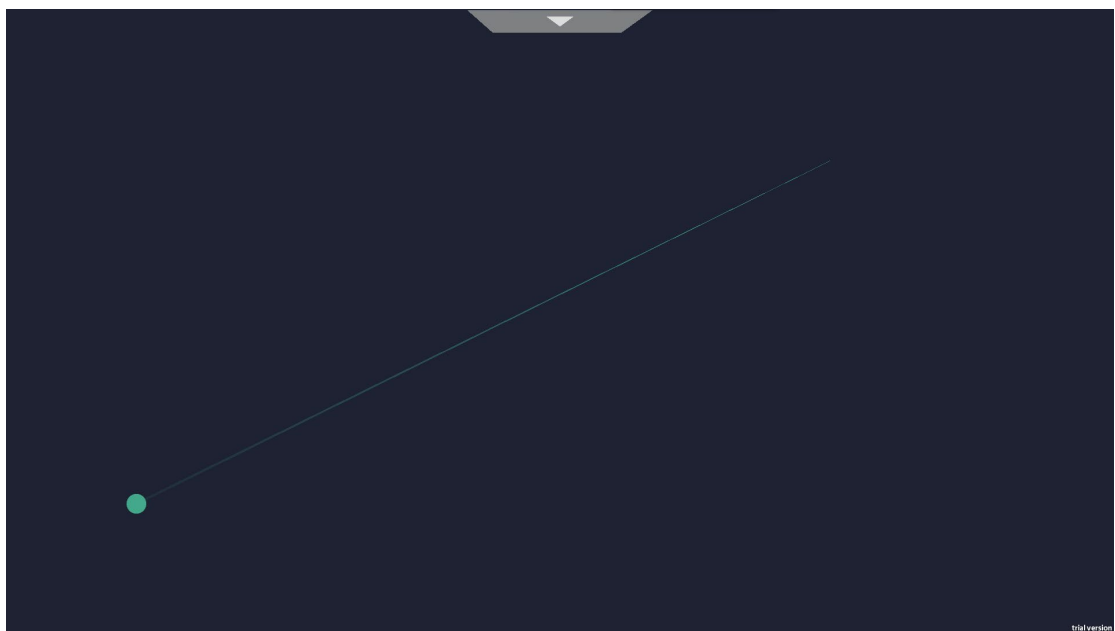
如上图所示 Width 随着移动尾部逐渐变细，颜色由本星球颜色逐渐过渡到白色。

渲染的效果很不错：



3.4.2 线条绘制

线条绘制与轨迹基本相同，主要作用是显示拖拽轨迹：



3.5 UI 制作、动画面板与背景音乐

3.5.1 功能面板的制作

功能面板会集成以上若干功能。功能面板的界面主要是由 Scroll View 实现的，并通过 Horizontal Layout Group 实现自动布局，以方便后续添加功能。



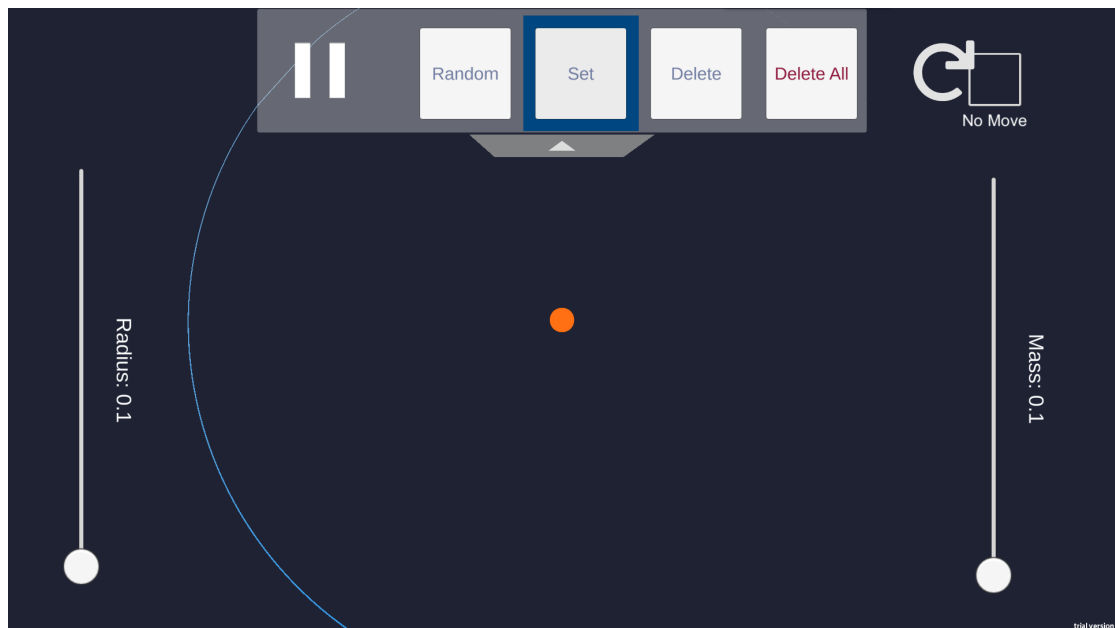
3.5.2 功能面板唯一选择

本游戏使用 Toggle Group，可以很方便的实现多个功能唯一选择。即选择另一个功能时上一个选择框会消失，新的选择框会生成。为每一个选项构建一个 Toggle 并绑定相同的 Toggle Group，即可统一管理。然后获取 Toggle Group 选中的对象即可知道当前的状态。

3.5.3 滑动条与选择框的设计

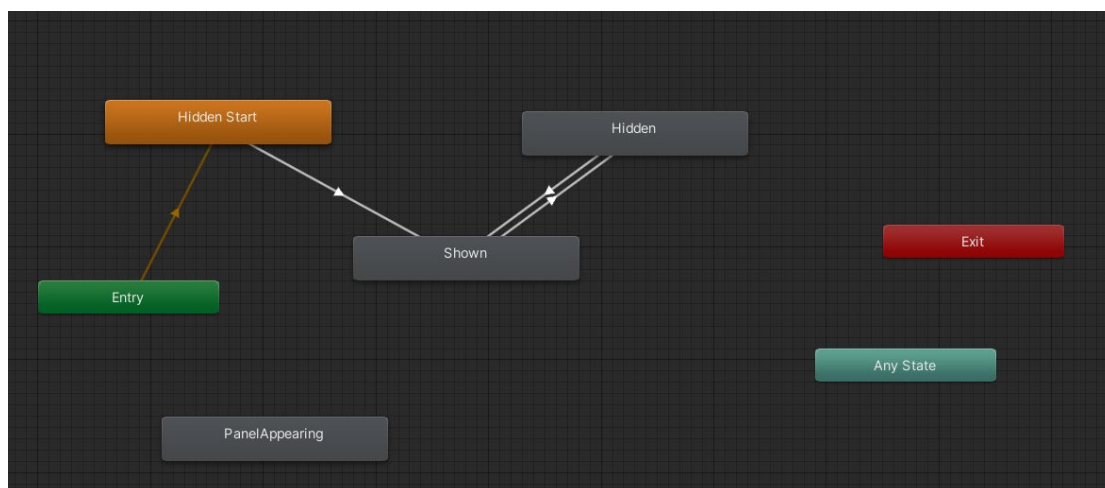
Set(设置)模式下，玩家可以滑动条来选择生成星球的半径与质量，通过设置 No Move 使得放置的星球静止不动。

滑动条采用的是 Unity 的 Slider 组件，拖动即可选择。No Move 采用的是 Toggle 组件，选中会生成一个方框。



3.5.4 动画面板

为了实现功能面板收拉与过度动画，这里采用了 Unity 的 Animator 动画机。动画机可以很方便的控制状态变化，有时候又叫状态机。本游戏涉及面板的状态有两个，出现与隐藏。状态机如下：



如上图，从 Entry 开始初始为隐藏状态，当拖拉板被点击时会切换到下一个 Shown 状态，再点击到 Hidden 状态，然后进入循环。

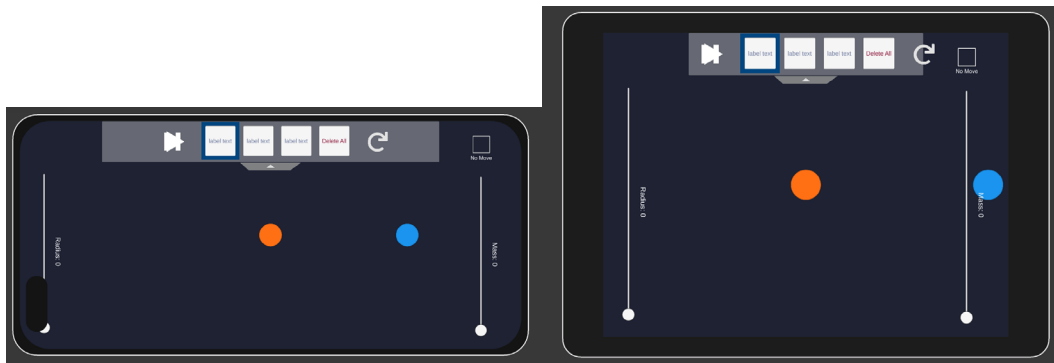
状态动画是通过 Animation 功能通过插入关键帧完成的。

3.5.5 背景音乐添加

Unity 中 AudioSource 组件用来控制 AudioClip（音频文件）的播放、暂停等操作。设置 AudioClip 初始默认开始播放、循环播放即可实现背景音乐的功能。同时与游戏暂停、开始绑定，即可实现游戏暂停时音乐也暂停。

3.6 多设备多平台界面适配

因为不同设备长宽不尽相同，因此若仅设置一个界面而不去考虑相对位置，则会遇到界面参差不齐的情况。因此需要对界面各个组件设置正确的锚点以适配多平台。



左边是 Huawei P40 Pro，而右边是 Apple ipad Air 2。可以发现做了多设备适配下界面会随着长宽自动变化，而不做的界面可能会凌乱。

四、总结

以上就是列举的大部分开发内容了，实际上并不是按照上述顺序、一步一步地做下去的，而是来回修改、重构，不断总结而成的。总结来说，之后游戏开发大致需要以下步骤会更好：

1. 确定目标、预期效果，先确定需要使用的基本技术栈；
2. 先实现底层的开发，例如事件、系统交互、UI，写好类封装，这样后面不至于回过头再看；
3. 先逐个实现基本的功能，而不考虑 UI；
4. 美化基本功能，重构部分代码；
5. 实现 UI、背景音乐等。

这样做是按照依存调用次数的顺序而来的，底层类、事件之后使用最多，调用依赖最多，应当好好考虑；而 UI、BGM 这些跟主体部分影响较小（解耦合度高），最后开发效果更佳。

相关链接与参考

[1] 本项目 GitHub 仓库 <https://github.com/164K/Universe>

[2] 背景音乐来源

Mirror <https://music.163.com/#/song?id=1449668208>

部分代码

EventClickToCreate.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics.Tracing;
using Unity.Mathematics;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.InputSystem;
using UnityEngine.UIElements;
using Random = UnityEngine.Random;
using SelectedMode = ModeManage.SelectedMode;

public class EventClickToCreate : MonoBehaviour
{
    public GameObject planet;
    public GameObject planetSet;

    public InputAction leftClick;

    public float massOrDensity = 1;
    public float radius = 1;
    // Random
    public bool isMassOrDensityRandom = false;
    public float modMin = 0.1f;
    public float modMax = 0.5f;

    public bool isRadiusRandom = false;
    public float radiusMin = 0.1f;
    public float radiusMax = 0.5f;

    public bool isColorRandom = false;
    public bool isNoMove = false;

    //new planet
    private GameObject newplanet;
    // Vector and its render
    private Vector2 cursorMove;
    private float3 oriPos;
```

```

private EventMessage emsg;
private ModeManage eventManage;
private UIMessagePipe messagePipe;
private LineRenderer lineRenderer;
private enum Status
{
    Origin, Creating
}
private Status status;

private void Awake()
{
    emsg = GetComponent<EventMessage>();
    eventManage = GetComponent<ModeManage>();
    lineRenderer = GetComponent<LineRenderer>();
    messagePipe = GetComponent<UIMessagePipe>();
    status = Status.Origin;
}
private void OnEnable()
{
    leftClick.Enable();
}
private void OnDisable()
{
    leftClick.Disable();
}

void Start()
{
    leftClick.started += LeftClick_started;
    leftClick.performed += LeftClick_performed;
}

private void Update()
{
    if (eventManage.CurrentMode == SelectedMode.Set ||
        eventManage.CurrentMode == SelectedMode.Random
    )
    {
        leftClick.Enable();
    }
    else if (status == Status.Origin)

```

```

        {
            leftClick.Disable();
        }
    }

private void LeftClick_started(InputAction.CallbackContext obj)
{
    if (!msg.IsOnUI) status = Status.Creating;
    if (status == Status.Creating)
    {
        OnCreatePlanetAtCursor();
        OnPlanetPaused();
        OnOtherStarted();
    }
}

private void LeftClick_performed(InputAction.CallbackContext obj)
{
    if (status == Status.Creating)
    {
        OnPlanetResume();
        OnOtherPerformed();
    }
    status = Status.Origin;
}

public void FixedUpdate()
{
    if (status==Status.Creating && leftClick.IsPressed())
        UpdateLines();
}

void UpdateLines()
{
    var endPos = GetCurrentCursorPosition();
    Vector3[] vertices = new Vector3[] { oriPos, endPos };
    lineRenderer.SetPositions(vertices);
}

void OnOtherStarted()
{
    oriPos = GetCurrentCursorPosition();
    lineRenderer.enabled = true;
}

```

```

void OnOtherPerformed()
{
    var finalPos = GetCurrentCursorPosition();
    cursorMove = new Vector2(finalPos.x - oriPos.x, finalPos.y - oriPos.y);
    lineRenderer.SetPositions(new Vector3[2] { oriPos, oriPos });
    lineRenderer.enabled = false;
    newplanet.GetComponent<Rigidbody2D>().velocity += cursorMove;
}

private void OnPlanetResume()
{
    GetComponent<Movement>().ResumeRigid();
}

private void OnPlanetPaused()
{
    GetComponent<Movement>().PauseRigid();
}

private void SetMassAndRadiusAndNoMove()
{
    if (eventManage.CurrentMode == SelectedMode.Set)
    {
        massOrDensity = messagePipe.MassFromSlider;
        radius = messagePipe.RadiusFromSlider;
        isNoMove = messagePipe.IsNoMove;
        newplanet.GetComponent<Planet>().IsNoMove = isNoMove;
    }
    else
    {
        if (isMassOrDensityRandom) massOrDensity = GetRandomBetween(modMin,
modMax);
        if (isRadiusRandom) radius = GetRandomBetween(radiusMin, radiusMax);
        //newplanet.GetComponent<Planet>().IsNoMove = isNoMove;
    }
}

private void OnCreatePlanetAtCursor()
{
    newplanet = Instantiate(planet, GetCurrentCursorPosition(),
Quaternion.identity, planetSet.transform);
    var sr = newplanet.GetComponent<SpriteRenderer>();
    SetMassAndRadiusAndNoMove();

    newplanet.GetComponent<Planet>().Initialized(massOrDensity, radius,

```

```

newplanet.transform.position, default, isColorRandom?GenerateRandomColor():default);

    // Set Trail Color
    newplanet.GetComponent<TrailRenderer>().startColor = sr.color;
    lineRenderer.startColor = ColorDarker(sr.color);
    lineRenderer.endColor = sr.color;
}

Color ColorDarker(Color color, float amout_rate = 0.3f)
{
    return color * amout_rate;
}

Color GenerateRandomColor()
{
    float r = Random.Range(0.0f, 1.0f);
    float g = Random.Range(0.0f, 1.0f);
    float b = Random.Range(0.0f, 1.0f);
    float a = 1.0f;
    return new Color(r, g, b, a);
}

float GetRandomBetween(float a, float b)
{
    return Random.Range(a, b);
}

Vector3 GetCurrentCursorPosition()
{
    return emsg.currentCursorWorldPosition;
}
}

```

Pauser.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Pauser : MonoBehaviour
{
    private Rigidbody2D rb;
    private Vector2 storedVelocity;
    private float storedAngularVelocity;
}

```

```

void Awake()
{
    rb = GetComponent<Rigidbody2D>();
    storedVelocity = rb.velocity;
    storedAngularVelocity = rb.angularVelocity;
}

public void Pause()
{
    if (rb.isKinematic) return;
    storedVelocity = rb.velocity;
    storedAngularVelocity = rb.angularVelocity;
    rb.isKinematic = true;
    rb.velocity = Vector2.zero;
    rb.angularVelocity = 0.0f;
}

public void Resume()
{
    if (!rb.isKinematic) return;
    rb.isKinematic = false;
    rb.velocity = storedVelocity;
    rb.angularVelocity = storedAngularVelocity;
}
}

```

Gravity.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using Unity.VisualScripting;
using UnityEngine;

public class Gravity : MonoBehaviour
{
    public float G = 1.0f;
    public struct MyType
    {
        public MyType(Transform t)
        {
            this.t = t;

```

```

        rb = t.GetComponent<Rigidbody2D>();
        pos = t.position;
    }

    public Transform t;
    public Rigidbody2D rb;
    public Vector3 pos;
}

// Start is called before the first frame update
void Start()
{
}

// Update is called once per frame
void FixedUpdate()
{
    List<MyType> children = new();
    List<GameObject> childrenGameObject =
GameObject.FindGameObjectsWithTag("Planet").ToList();
    childrenGameObject.ForEach((GameObject g) => children.Add(new
MyType(g.transform)));

    foreach (var now_t in children)
    {
        var totalForce = new Vector3();
        foreach (var another_t in children)
        {
            if (now_t.t != another_t.t)
            {
                var r = another_t.pos - now_t.pos;
                var r_scale = r.magnitude;
                totalForce += G * now_t.rb.mass * another_t.rb.mass / (r_scale *
r_scale) * r.normalized;
            }
        }
        now_t.rb.AddForce(totalForce);
    }
}
}

```

Planet.cs

```

using System;
using System.Collections;

```

```

using System.Collections.Generic;
using System.IO;
using Unity.VisualScripting;
using UnityEngine;

public struct PlanetAttributes
{
    public PlanetAttributes(bool isNoMove, Vector2 velocity, Vector2 position, float
radius, float mass, Color planetColor, Color planetTrailColor)
    {
        this.isNoMove = isNoMove;
        this.velocity = velocity;
        this.position = position;
        this.radius = radius;
        this.mass = mass;
        this.planetColor = planetColor;
        this.planetTrailColor = planetTrailColor;
    }

    bool isNoMove;
    Vector2 velocity;
    Vector2 position;
    float radius;
    float mass;
    Color planetColor;
    Color planetTrailColor;
}

public class Planet : MonoBehaviour
{
    Rigidbody2D rb;
    CircleCollider2D cc;
    SpriteRenderer sr;

    private bool isNoMove = false;
    private Vector2 velocity;
    private Vector2 position;
    private float radius;
    private float mass;
    private Color planetColor;
    private Color planetTrailColor;

    public PlanetAttributes GetAttributes()
    {
        return new PlanetAttributes(isNoMove, velocity, position, radius, mass,

```



```

planetColor, planetTrailColor);
    }

    private TrailRenderer trailRenderer;

    public float Mass
    {
        set { mass = value; if (rb.useAutoMass) cc.density = value; else rb.mass =
value; }
        get { return (rb.useAutoMass) ? cc.density : rb.mass; }
    }
    public float Radius
    {
        set { radius = value; transform.localScale = (value * 2) * Vector3.one; }
        get { return transform.localScale.x; }
    }
    public Vector2 Position
    {
        set { position = value; rb.MovePosition(value); }
        get { return transform.position; }
    }
    public Vector2 Velocity
    {
        set { velocity = value; rb.velocity = value; }
        get { return rb.velocity; }
    }
    public Color PlanetColor
    {
        set { planetColor = value; sr.color = value; }
        get { return sr.color; }
    }
    public Color TrailColor
    {
        set { trailRenderer.startColor = ColorDarker(value); trailRenderer.startColor =
value; }
    }

    public bool IsKinematic
    {
        set { rb.isKinematic = value; }
        get { return rb.isKinematic; }
    }
    public bool IsNoMove

```

```

{
    set { isNoMove = value; }
    get { return isNoMove; }
}

Color ColorDarker(Color color, float amout_rate = 0.3f)
{
    return color * amout_rate;
}

private void Awake()
{
    rb = GetComponent<Rigidbody2D>();
    cc = GetComponent<CircleCollider2D>();
    sr = GetComponent<SpriteRenderer>();
    trailRenderer = GetComponent<TrailRenderer>();
}

public void Initialized(float massOrDensity, float radius, Vector2 position =
default, Vector2 velocity = default, Color color=default)
{
    Velocity = velocity;
    Radius = radius;
    Position = position;
    Mass = massOrDensity;
    PlanetColor = color;
    TrailColor = color;
}

private void FixedUpdate()
{
    if (IsNoMove)
    {
        IsKinematic = true;
        Velocity = Vector2.zero;
    }
}
}

```