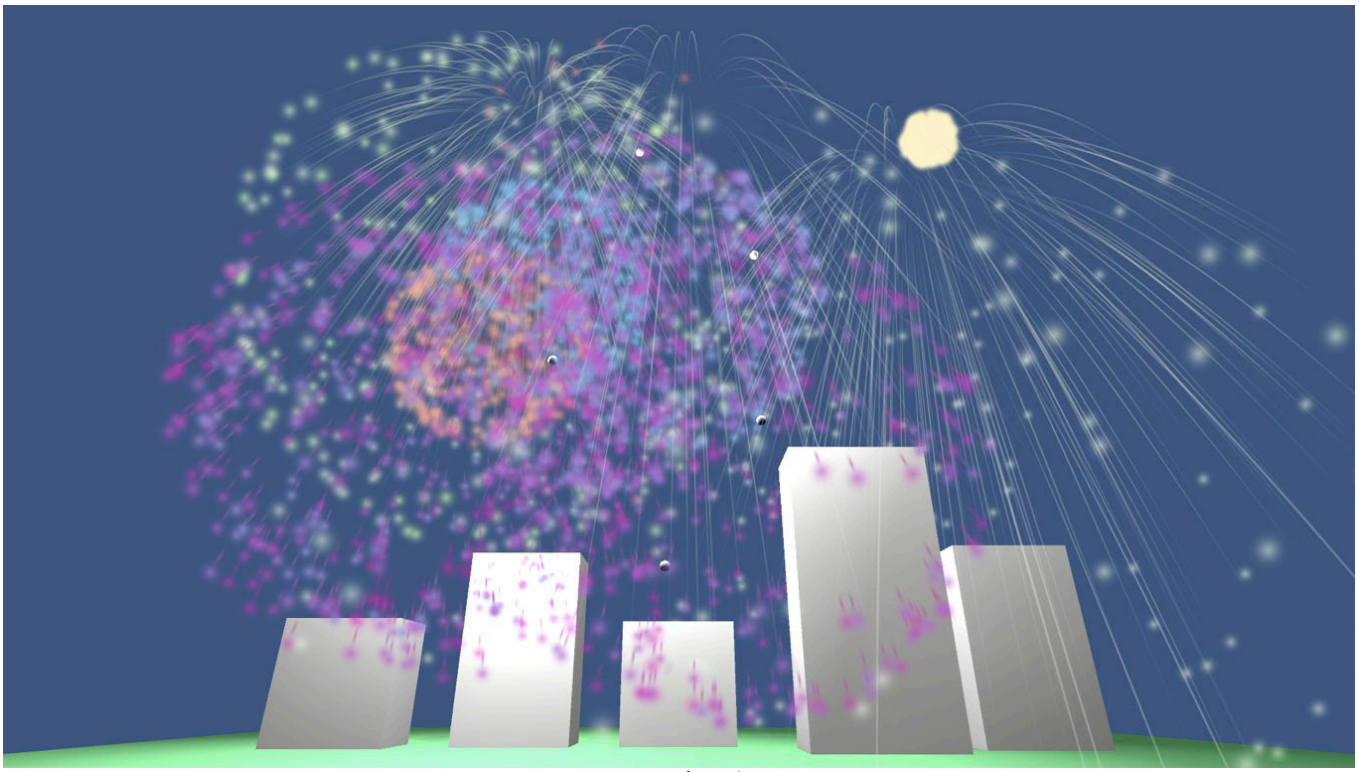
ゲームプログラミング

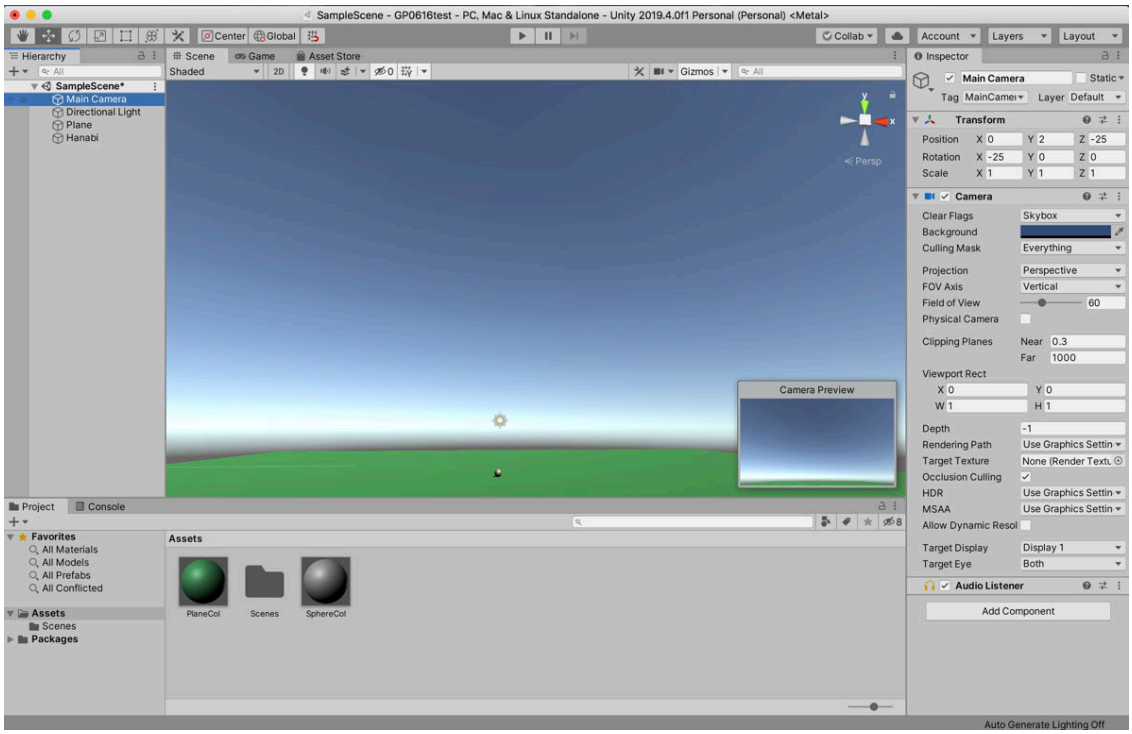
第8回　水野慎士

今日の目標：花火



・エフェクトオブジェクトの使用
・Prefabの作成と使用

シーン作成



Plane
Pos:(0.0, 0.0, 0.0)
Rot:(0.0, 0.0, 0.0)
Scl:(5.0, 5.0, 5.0)

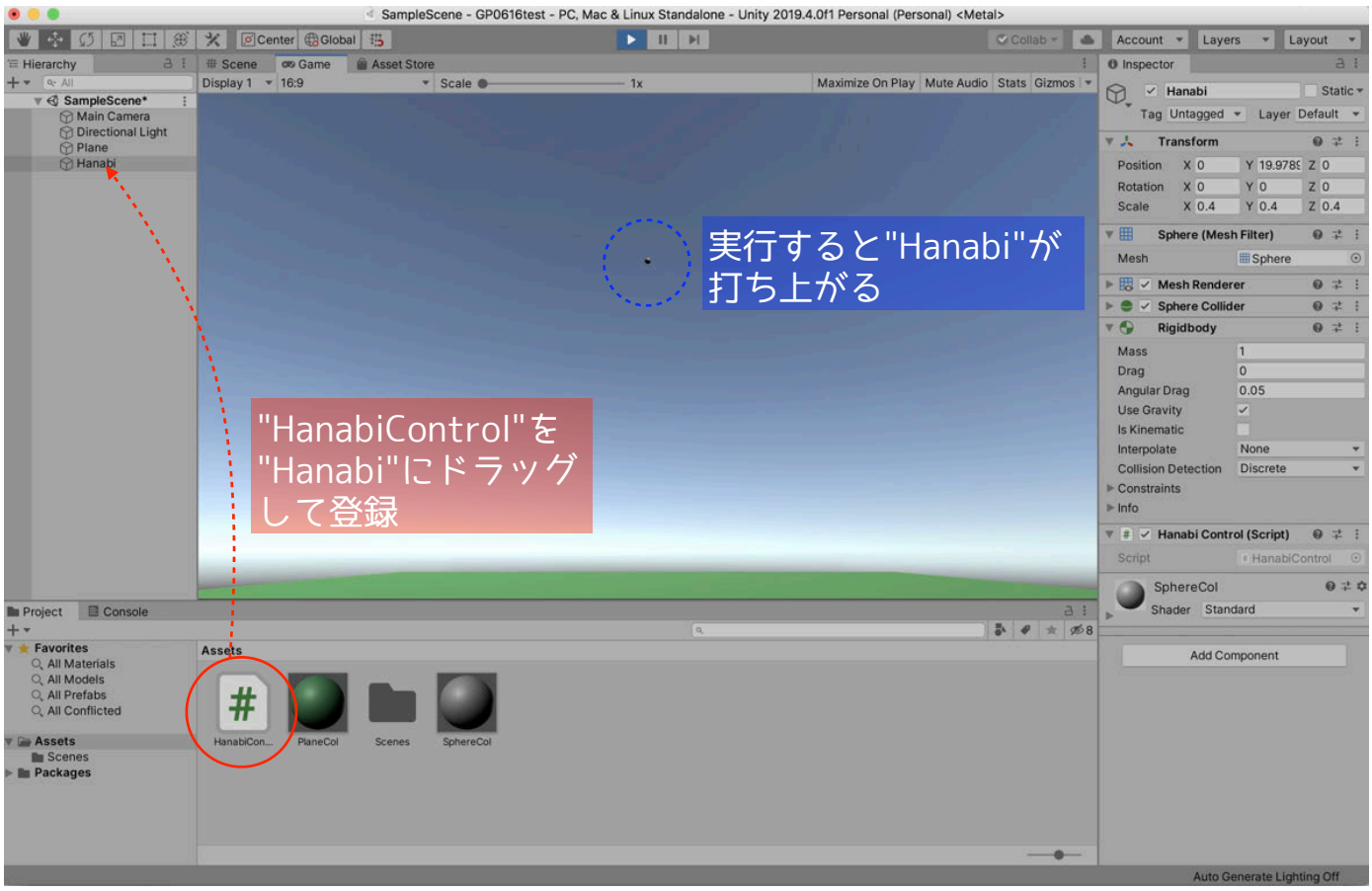
Sphere→名称"Hanabi"→Rigidbody追加
Pos:(0.0, 0.2, 0.0)
Rot:(0.0, 0.0, 0.0)
Scl:(0.4, 0.4, 0.4)

"HanabiControl.cs"の作成と記述

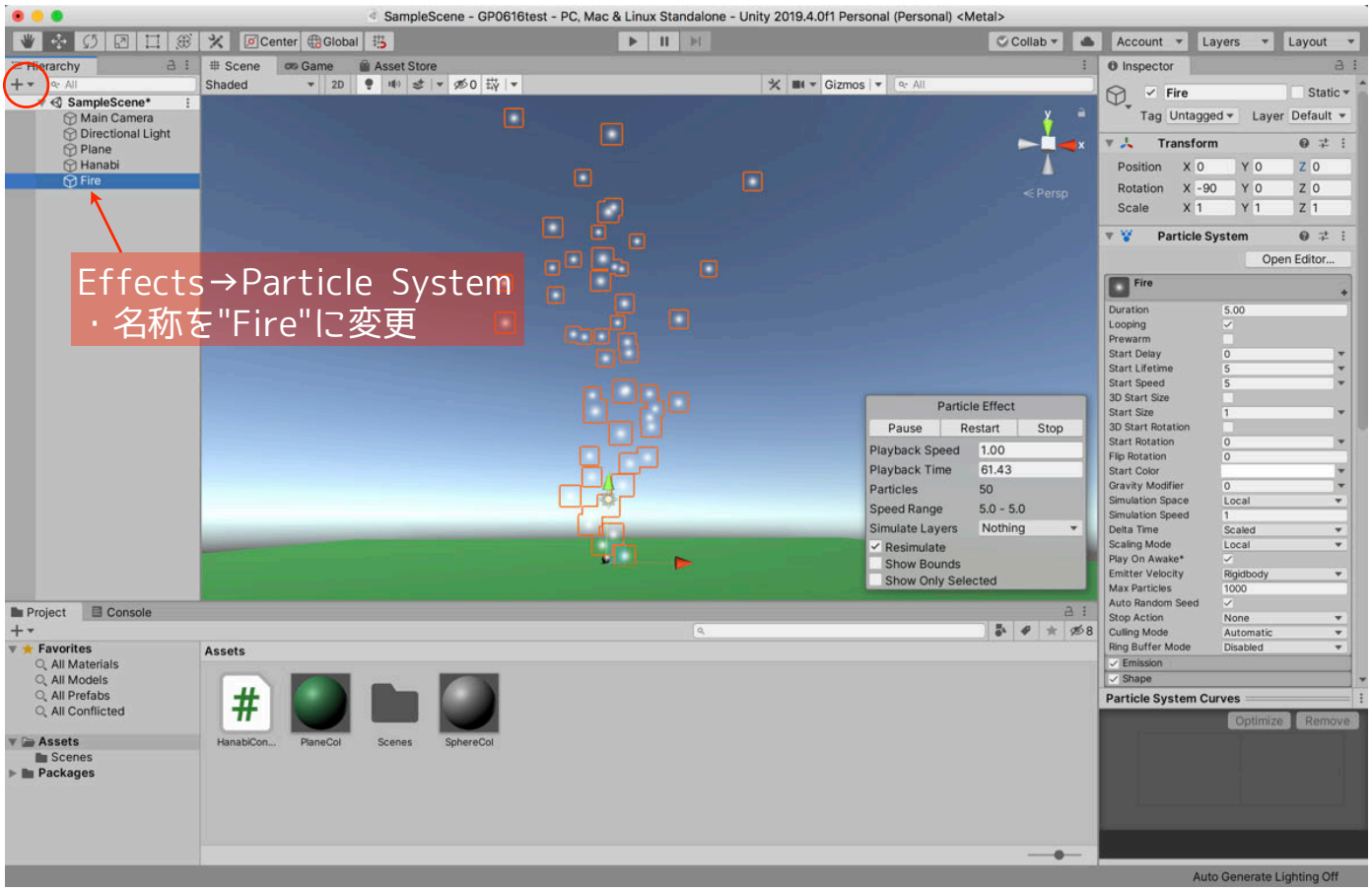
HanabiControl.cs
HanabiControl ▸ Start()

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class HanabiControl : MonoBehaviour
6   {
7       // Start is called before the first frame update
8       void Start()
9       {
10          //自分自身のRigidbodyコンポーネントに(0,1000,0)の力を加える
11          GetComponent<Rigidbody>().AddForce(0, 1000, 0);
12      }
13
14      // Update is called once per frame
15      void Update()
16      {
17
18      }
19  }
20
```

"HanabiControl.cs"の"Hanabi"への登録と実行

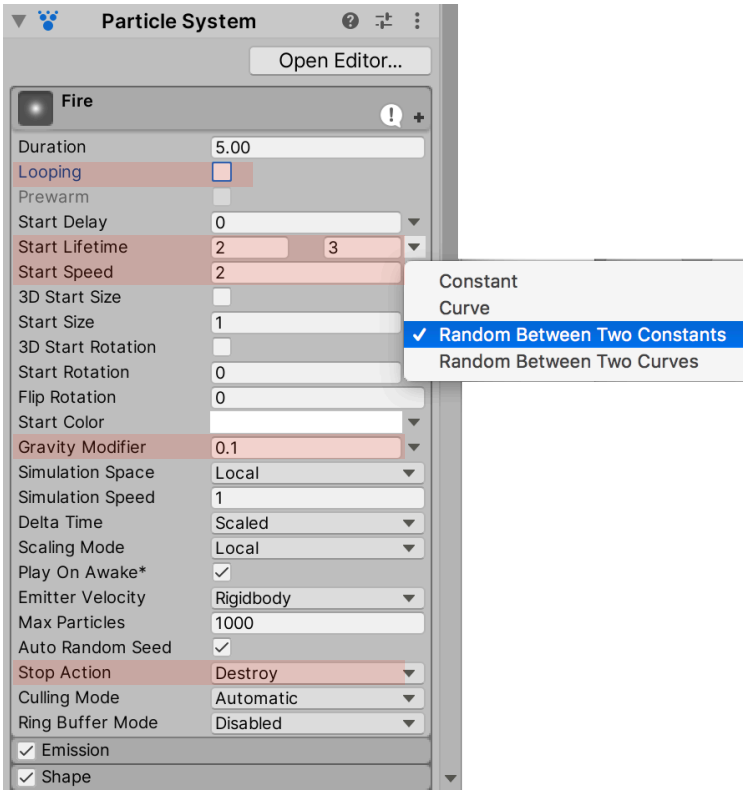


Particle System オブジェクト追加

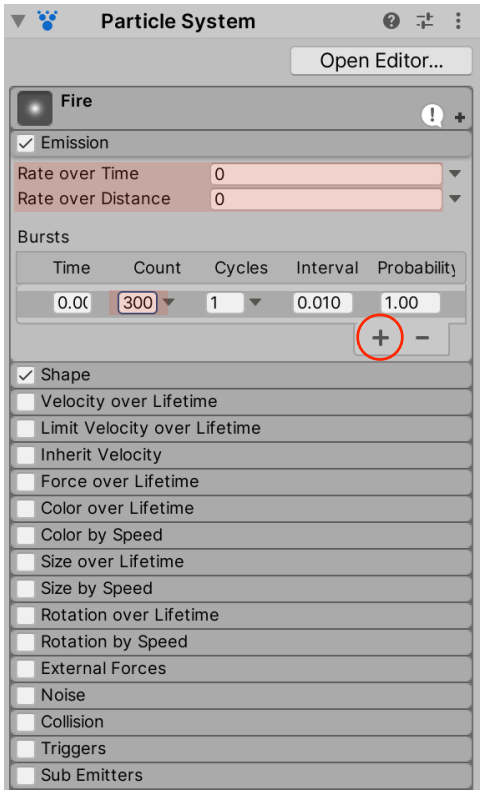


"Fire"の設定

基本設定

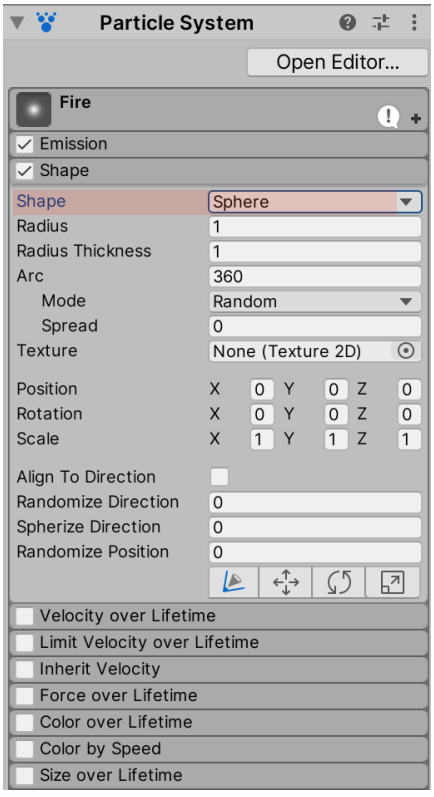


Emission

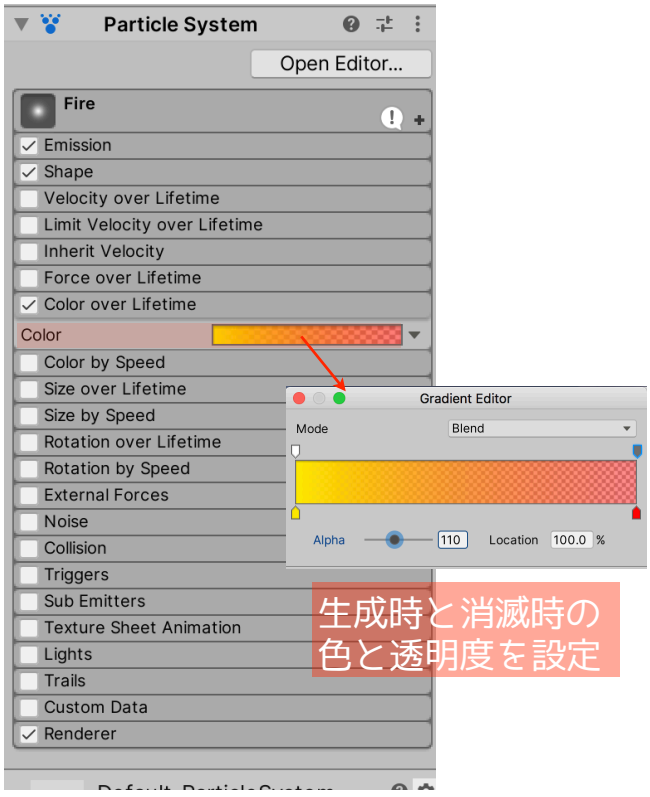


"Fire"の設定

Sphere



Color over Lifetime

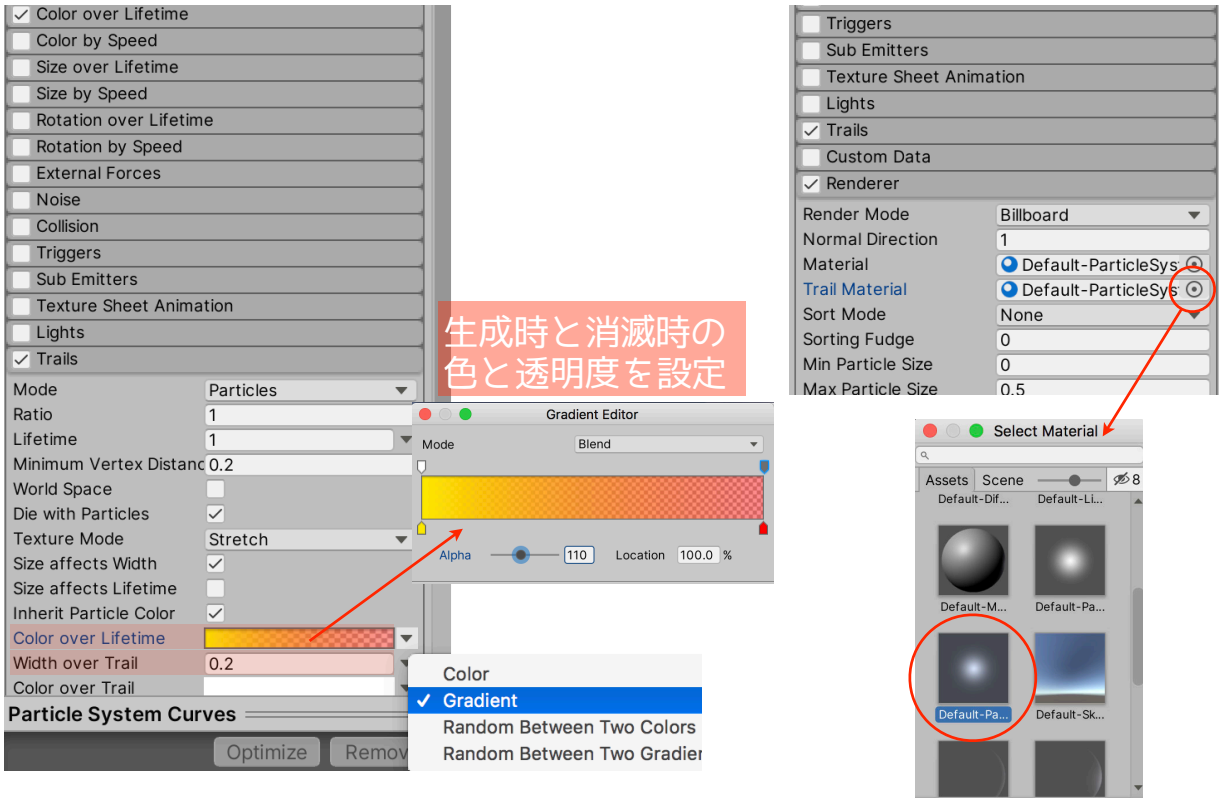


生成時と消滅時の
色と透明度を設定

"Fire"の設定

Trails

Renderer

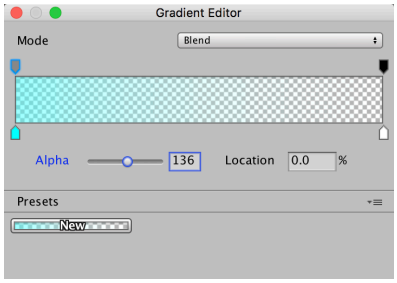


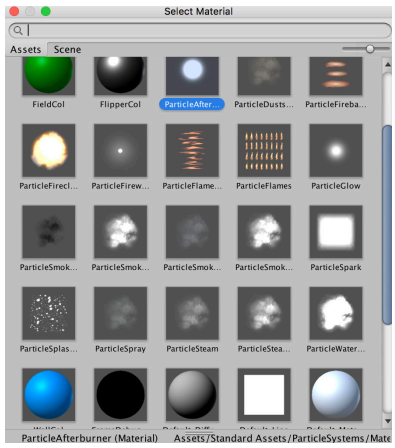
主なパーティクルデフォルトモジュール

- Duration：実行時間の長さ
- Looping：繰り返し処理のオンオフ
- Prewarm：発生済み表現のオンオフ
- Start Delay：実行待機時間
- Start Lifetime：寿命
- Start Speed：初期速度
- Start Rotation：初期角度
- Start Color：初期色
- Simulation Space：生成空間
- Play On Awake：起動時放出のオンオフ
- Max Particles：同時存在個数の上限値

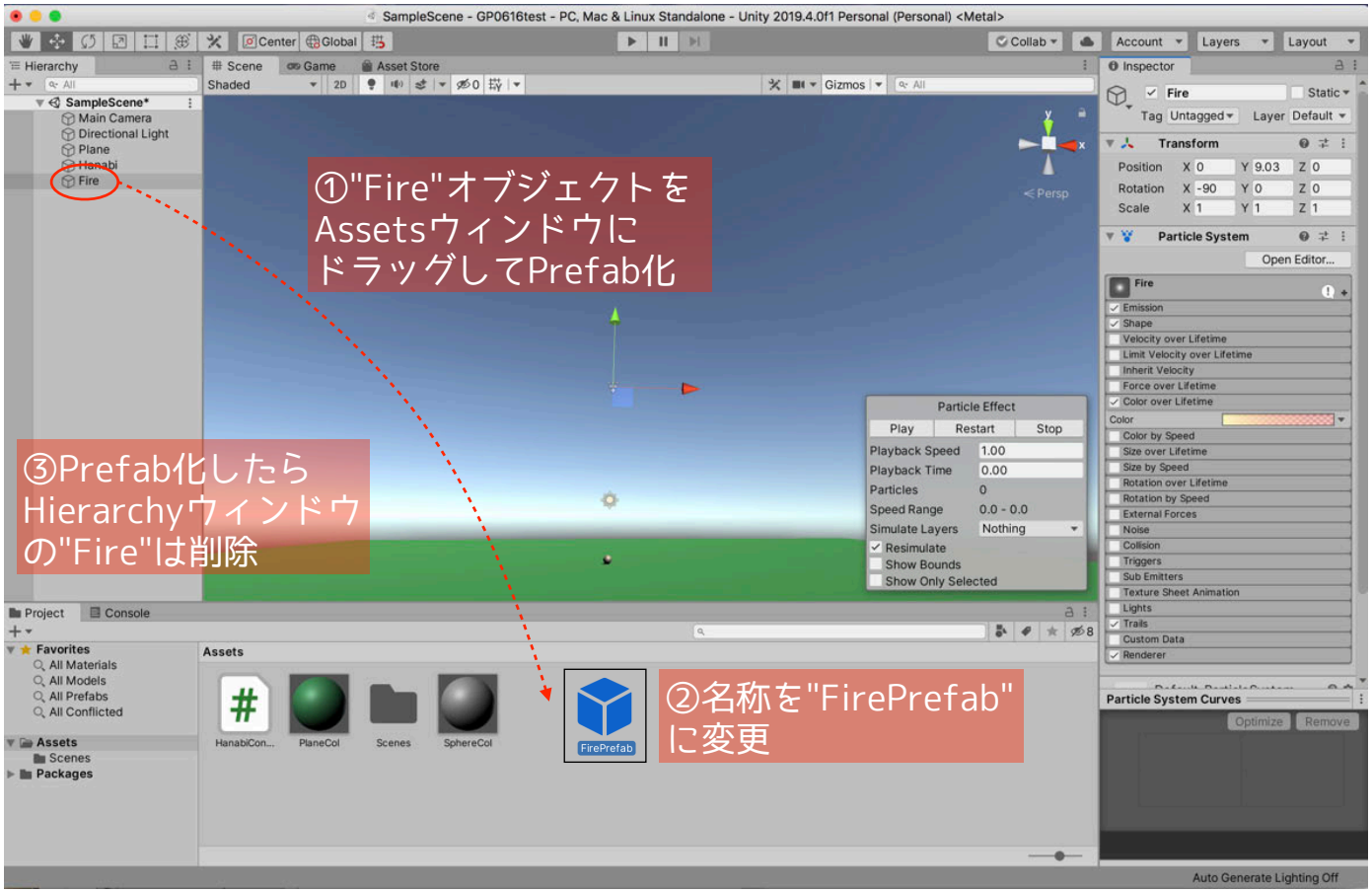
その他の主なモジュール

- Emission：放出頻度
  - Rate over Time：時間に対する密度
  - Rate over Distance：距離(速度)に対する密度
- Shape：放出形状
  - Sphere(球)，Cone(円錐)など
- Velocity over Lifetime：各パーティクル速度
- Color over Lifetime：色変化
- Renderer：描画方法
  - Material：使用マテリアル
    - これを指定しないとパーティクルが紫色の四角形で表示される





"Fire"オブジェクトのPrefab化



"HanabiControl.cs"への追加記述

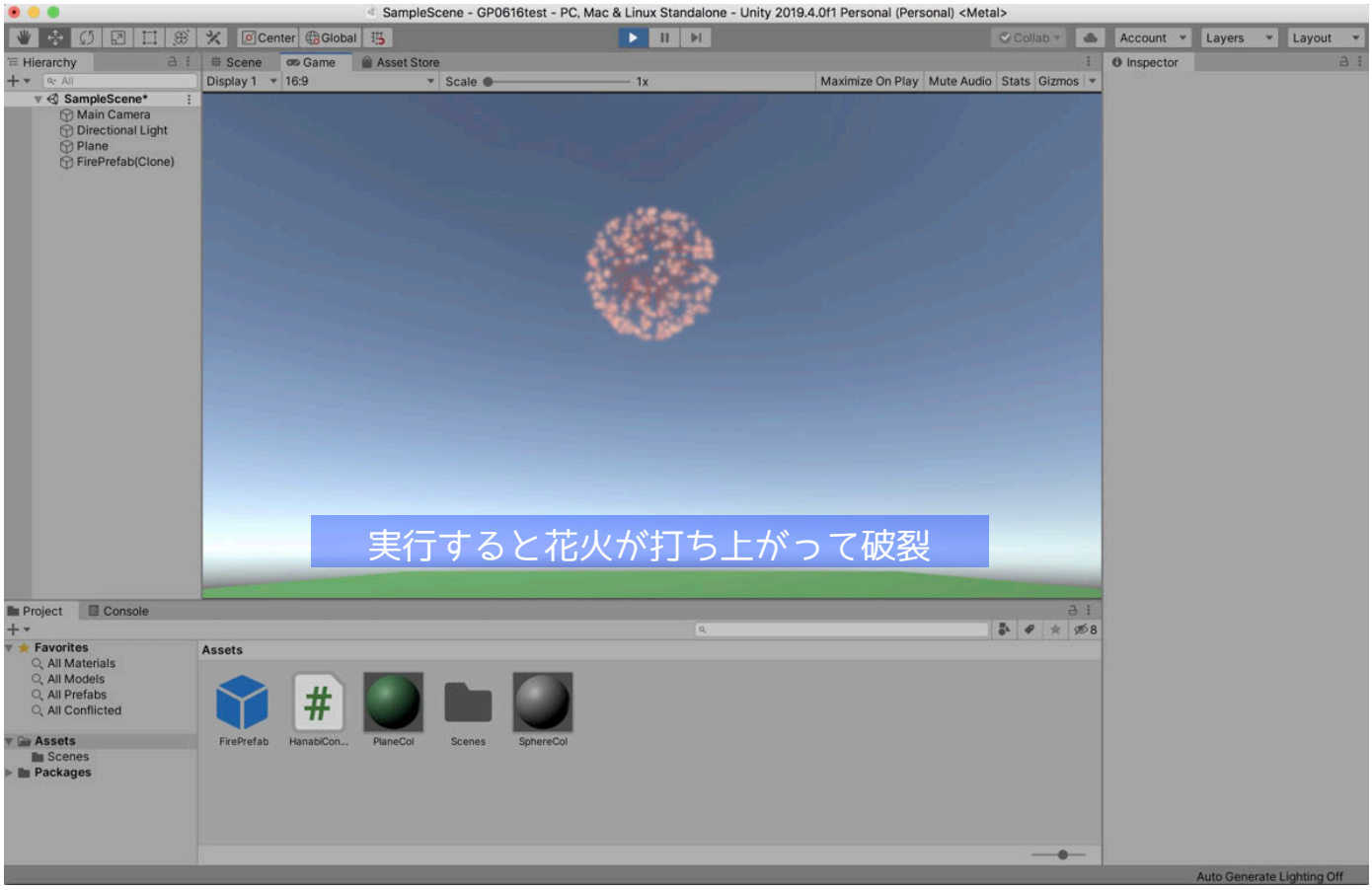
HanabiControl.cs

HanabiControl ▸ Update()

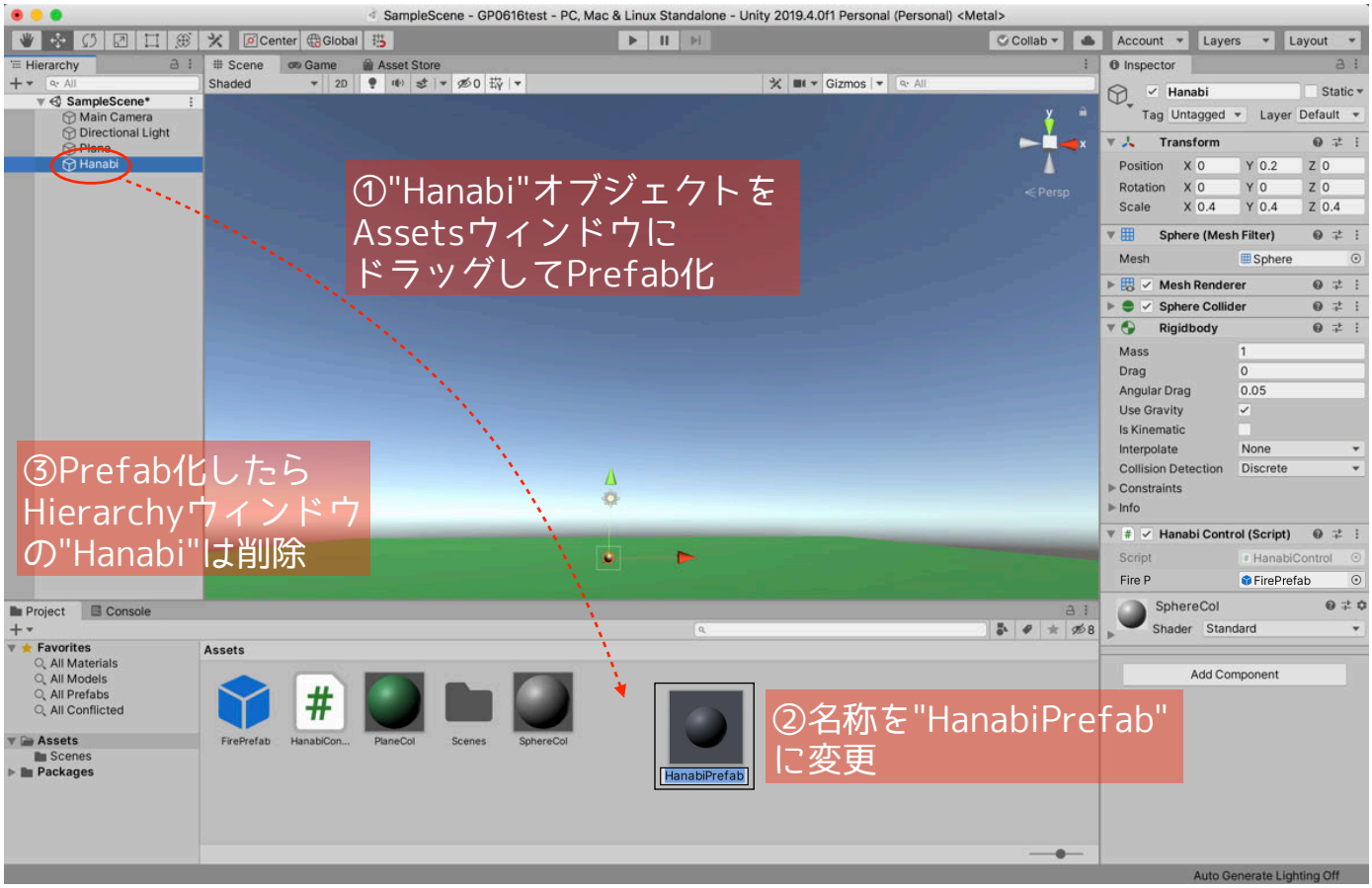
```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class HanabiControl : MonoBehaviour
6   {
7       public GameObject fireP;  //Fire Prefab格納用
8
9       // Start is called before the first frame update
10      void Start()
11      {
12          //自分自身のRigidbodyコンポーネントに(0,1000,0)の力を加える
13          GetComponent<Rigidbody>().AddForce(0, 1000, 0);
14      }
15
16      // Update is called once per frame
17      void Update()
18      {
19          //自分自身のy方向速度が0未満のとき
20          if (GetComponent<Rigidbody>().velocity.y<0)
21          {
22              //"fireP"を実体化して"fire"を生成
23              GameObject fire = Instantiate(fireP) as GameObject;
24              //"fire"の位置を自分自身と同じにする
25              fire.transform.position = transform.position;
26              //自分自身を破棄する
27              Destroy(gameObject);
28          }
29      }
30  }
```

"FirePrefab"の"Hanabi"オブジェクトへの登録

実行画面



"Hanabi"オブジェクトのPrefab化



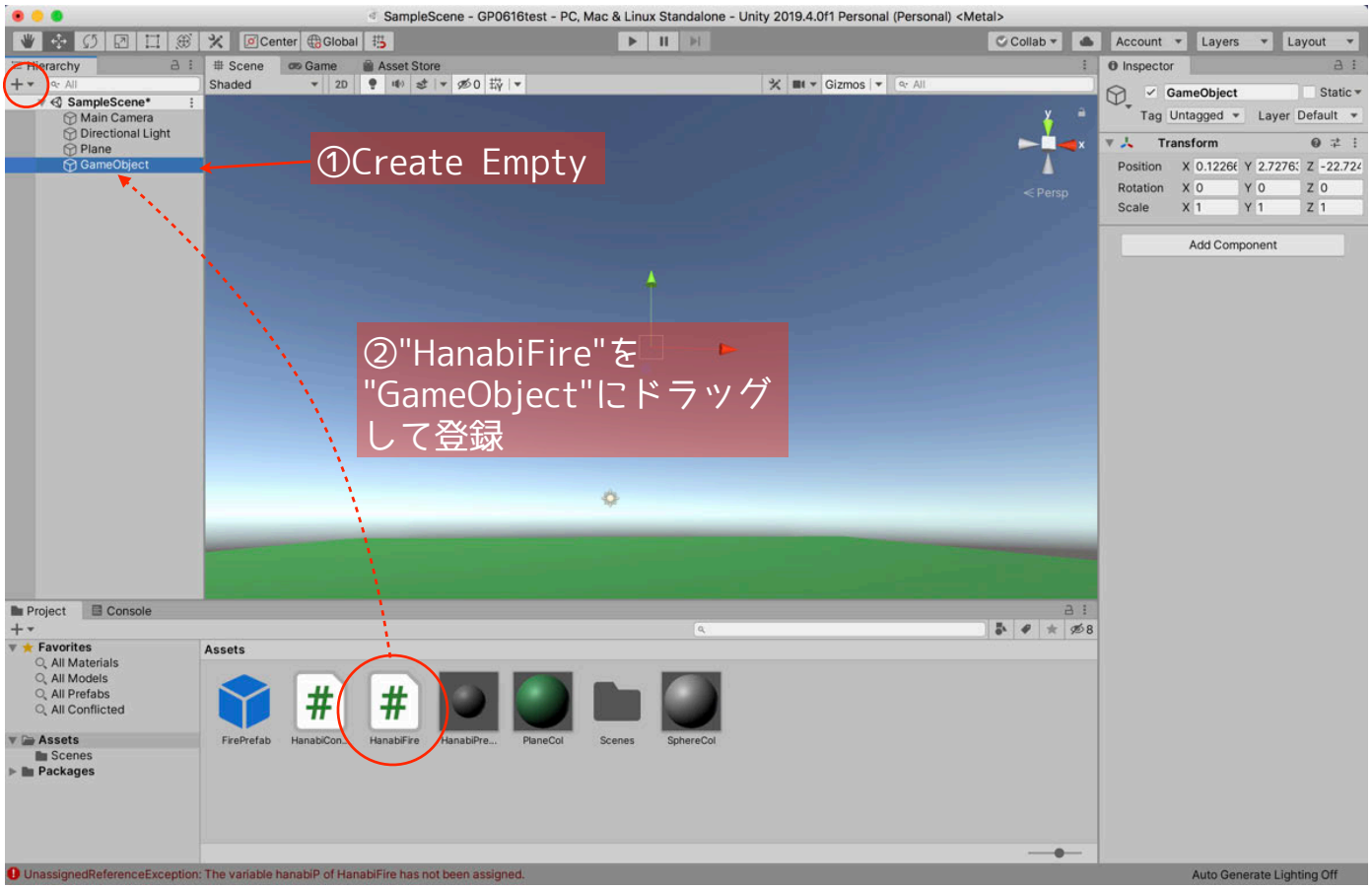
"HanabiFire.cs"の作成と記述

HanabiControl.cs　HanabiFire.cs

HanabiFire ▸ Update()

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class HanabiFire : MonoBehaviour
6   {
7       public GameObject hanabiP;  //Hanabi Prefab格納用
8
9       // Start is called before the first frame update
10      void Start()
11      {
12
13      }
14
15      // Update is called once per frame
16      void Update()
17      {
18          //キーボードのスペースキーが押されたとき
19          if (Input.GetKeyDown(KeyCode.Space))
20          {
21              //"hanabiP"を実体化して"hanabi"を生成
22              GameObject hanabi = Instantiate(hanabiP) as GameObject;
23          }
24      }
25  }
26
```

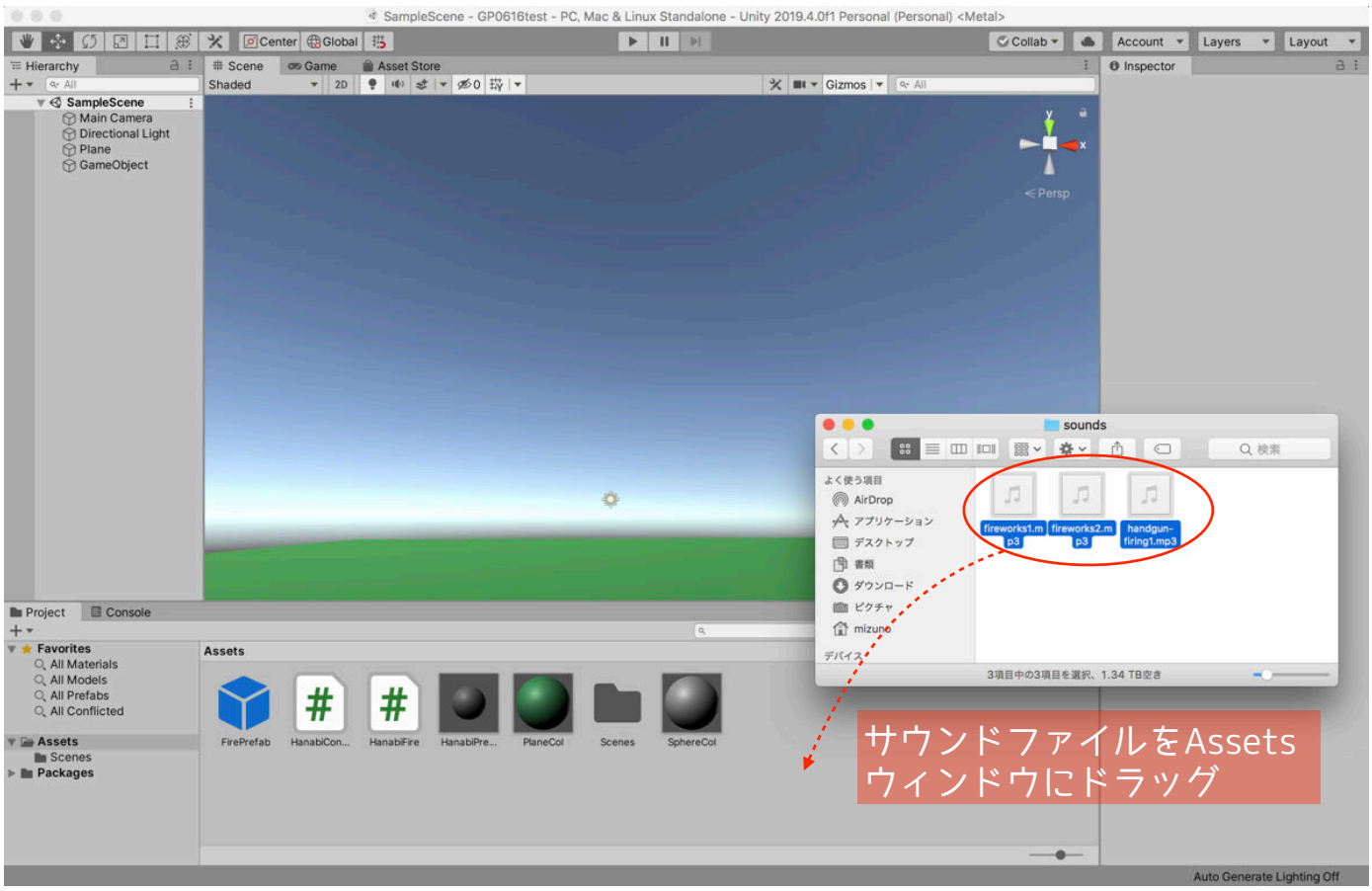
空オブジェクトの作成と"HanabiFire.cs"の登録



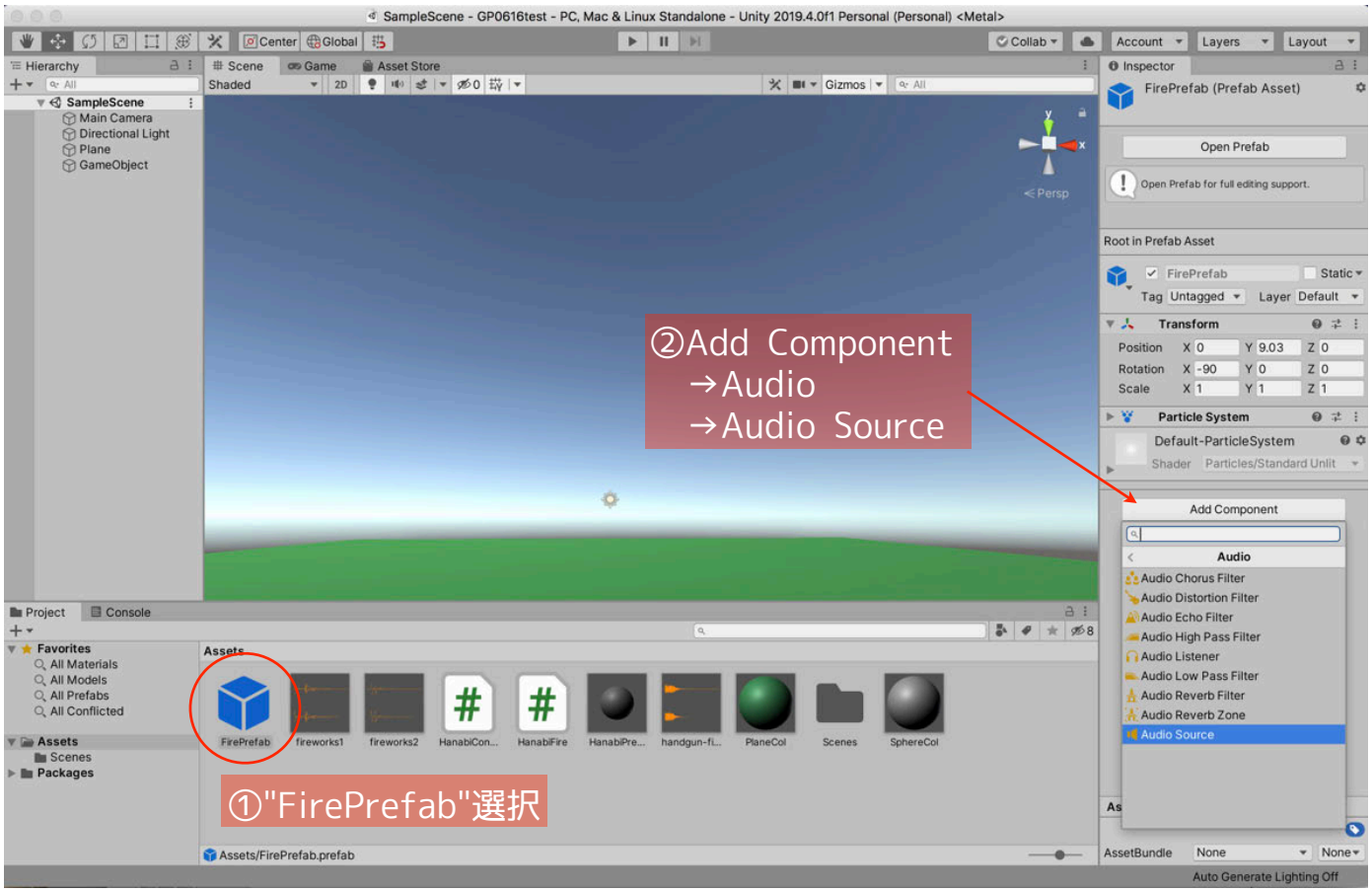
"HanabiPrefab"の"GameObject"オブジェクトへの登録

実行画面

サウンドファイルの追加



"FirePrefab"へのAudio Sourceコンポーネント追加



"FirePrefab"へのAudioClip登録

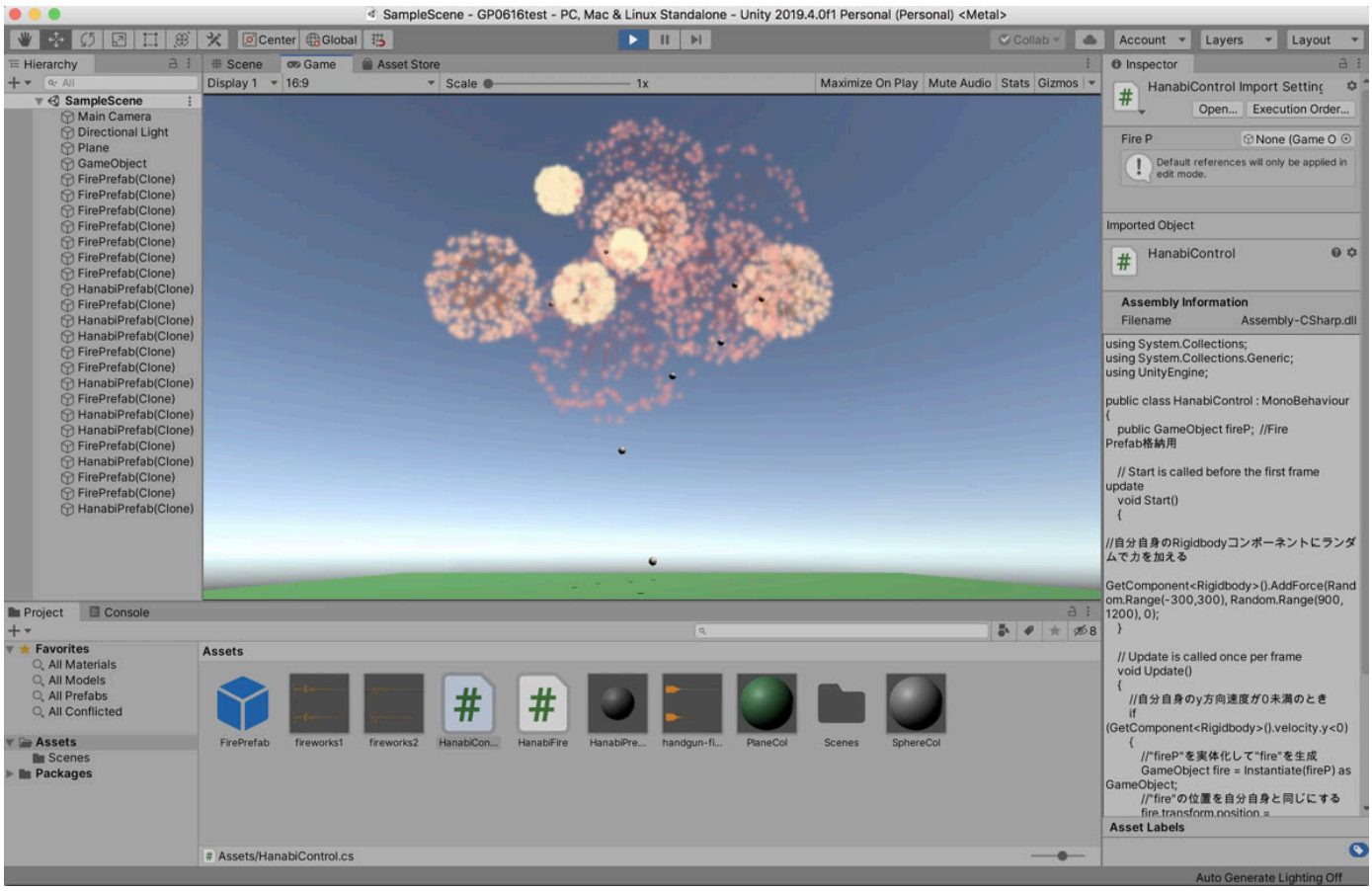
"HanabiControl.cs"の修正

HanabiControl.cs　HanabiFire.cs

HanabiControl ▸ Start()

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class HanabiControl : MonoBehaviour
6   {
7       public GameObject fireP;  //Fire Prefab格納用
8
9       // Start is called before the first frame update
10      void Start()
11      {
12          //自分自身のRigidbodyコンポーネントにランダムで力を加える
13          GetComponent<Rigidbody>().AddForce(Random.Range(-300,300), Random.Range(900, 1200), 0);
14      }
15
16      // Update is called once per frame
17      void Update()
18      {
19          //自分自身のy方向速度が0未満のとき
20          if (GetComponent<Rigidbody>().velocity.y<0)
21          {
22              //"fireP"を実体化して"fire"を生成
23              GameObject fire = Instantiate(fireP) as GameObject;
24              //"fire"の位置を自分自身と同じにする
25              fire.transform.position = transform.position;
26              //自分自身を破棄する
27              Destroy(gameObject);
28          }
29      }
30  }
31
```

実行画面



多彩な花火のための工夫

- 花火の種類を増やすには
  - "FirePrefab"を複製して，Inspectorでパラメータ変更
  - "HanabiControl.cs"中の"fireP"の配列化
    - `public GameObject[] fireP;`
  - "HanabiPrefab"のInspector中の"Fire P"欄が複数個になるので，複製した"FirePrefab"を登録
  - "HanabiControl.cs"中の"fireP"の実体化の部分を複製個数に応じて修正
    - `GameObject fire =Instantiate(fireP[Random.Range(0,3)]) as GameObject;`

多彩な花火のための工夫

- 花火による発光を再現するには
  - PointLightオブジェクト作成
    - Hierarchy→Light→Point Light
  - Range(到達範囲)とIntensity(強度)を調整
  - Intensityを少しずつ下げながら，しきい値を下回った場合に自分自身を破棄するC#スクリプトを作成
    →作成したPointLightオブジェクトに登録
  - PointLightオブジェクトをPrefab化
  - "HanabiControl.cs"にPointLight Prefabを実体化する記述を追加
    - "FireP"の実体化の記述を参考に

課題：派手な花火の打ち上げ動画を生成しよう