```
// HalfPoint2.cs

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

public class HalfPoint2 : MonoBehaviour
{
    [SerializeField]
    private GameObject m_FireCharacter;                         // 火文字
    [SerializeField]
    private GameObject m_FireCharacterCamera;                   // 火文字を映すためのカメラ
    [SerializeField]
    private GameObject m_FireworksCamera;                       // 花火を映すためのカメラ
    [SerializeField]
    private Transform m_CameraTarget;                           // 花火の打ち上がった頂点（カメラが回転し向く方向）
    [SerializeField]
    private FadeInOutController m_FadeController;              // フェードスクリプト
    [SerializeField]
    private float m_RotateSpeed;                                // 回転スピード（カメラの回転スピード）
    [SerializeField]
    private Vector3 m_MainCameraEuler;                          // メインカメラが回転する前の回転角度を保持する変数（オイラー角）
    [SerializeField]
    private List<Tourou> m_TourouList = new List<Tourou>();             // 全体の灯篭リスト

    [HideInInspector]
    public bool m_TriggerEnterIn = false;                       // トリガー内に進入してきたかどうか
    [HideInInspector]
    public bool m_CameraIsScreen = false;                       // ターゲット方向に回転するためのフラグ
    [HideInInspector]
```

```
        public bool m_RotateCompletion;                 // TargetLookRotateAngleメソッドの引数
        [HideInInspector]
        public bool m_FireworksFadeFlag;                // 花火用のフェードフラグ
        [HideInInspector]
        public bool m_EndFireworksFadeFlag;             // 花火終了のフェードフラグ
        [HideInInspector]
        public bool m_FireworksEndingStart;             // 花火終了演出スタート
        [HideInInspector]
        public bool m_FireworksEndingProduction;        // 花火終了演出フラグ
        [HideInInspector]
        public bool m_ReloadListFlag = true;            // リスタート時等の時にリストに追加し設定するためのフラグ

        public int m_TourouCount = 0;                   // HalfPoint2に進入してきた灯篭の数
        private int m_ArriveTourouIndex;
        private bool m_TourouEnterListFlag = false;     // 沈んでいない灯篭を常にリスト管理するためのフラグ
        private bool m_FireCharacterFadeFlag;           // 火文字用のフェードフラグ
        private bool m_FireProductionTimeFlag;          // 火文字演出時間用のフラグ
        private bool m_FireCharacterProduction;         // 火文字演出フラグ
        private bool m_FireworksProduction;             // 花火演出フラグ
        private bool m_DonotReachTourouFlag;            // HalfPoint2に一番最初に触れた灯篭フラグ
        private bool m_WithinCoolTimerFlag;             // クールタイマー内フラグ
        private bool m_Production;

        [SerializeField]
        private float m_CoolTimer;                      // クールタイマー
        [SerializeField]
        private float m_EnterInCoolTime;                // HalfPoint2に触れていない灯篭を待つ時間
        private bool m_CoolTimerFlag;                   // クールタイマー用のフラグ
```

```
    // Use this for initialization
    void Start ()
    {
        // FadeInOutControllerを取得
        m_FadeController =
GameObject.FindGameObjectWithTag("Fade").GetComponent<FadeInOutController>();

        // 火文字（パーティクル）オブジェクトを非表示にする
        m_FireCharacter.SetActive(false);
        m_FireCharacterCamera.SetActive(false);
        m_FireworksCamera.SetActive(false);
    }

    private void OnEnable()
    {
        if (m_ReloadListFlag)
        {
            // 沈んでない灯篭をリストに追加（灯篭全体のリスト）
            foreach (GameObject tourou in GameObject.FindGameObjectsWithTag("Tourou"))
            {
                Tourou tourouScript = tourou.GetComponent<Tourou>();
                if (tourouScript.tourouDown) continue;
                m_TourouList.Add(tourouScript);

                m_ReloadListFlag = false;
            }
        }
    }

    // Update is called once per frame
    void Update ()
    {
        //TourouConflicted();

        // 沈んでいる灯篭をリストから削除
        foreach (GameObject tourou in GameObject.FindGameObjectsWithTag("Tourou"))
```

```
        {
            Tourou tourouScript = tourou.GetComponent<Tourou>();
            if (tourouScript.tourouDown == false) continue;
            m_TourouList.Remove(tourouScript);
        }

        if (m_TourouEnterListFlag == true)
        {
            OnTriggerEnterConflicted();
            DoNotReachTheMidpoint();
        }

        if (m_FireCharacterProduction == true)
        {
            FireCharacterFadeProcessing();
        }

        if (m_FireworksProduction == true)
        {
            StartFireworksFadeProcessing();
        }

        if (m_FireworksEndingProduction == true)
        {
            EndFirewoorksFadeProcessing();
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.tag == "Tourou")
        {
            // トリガーに触れたか
            m_TriggerEnterIn = true;
            // HalfPoint2に進入してきた灯篭の数
            m_TourouCount = m_TourouCount + 1;
```

```
                // 回転する前に回転角度を変数を用いて保持させる　※ のちにカメラを元の回転角度に戻すため
                //m_MainCameraEuler = Camera.main.transform.eulerAngles;
                m_TourouEnterListFlag = true;
            }
        }

        /// <summary>
        /// トリガーに触れた灯篭
        /// </summary>
        private void OnTriggerEnterConflicted()
        {
            // トリガーに触れた灯篭を検知
            if (m_TriggerEnterIn)
            {
                IEnumerable<Tourou> tourous = m_TourouList.Where(tourou => tourou.tourouDown == false && tourou.m_TriggerEnterIn == true);

                // 沈んでいない灯篭とトリガーに衝突してきた灯篭の数が同じ数なら
                if (m_TourouList.Count == tourous.Count(tourou => tourou.tourouDown == false && tourou.m_TriggerEnterIn == true))
                {
                    m_CoolTimer += Time.deltaTime;

                    if (m_CoolTimer >= 5.0f)
                    {
                        m_CoolTimer = 0.0f;
                        // カメラの追従を止める

GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Chasing_Camera>().enabled = false;
                        // カメラの回転フラグ
                        m_CameraIsScreen = true;
                        m_FireCharacterProduction = true;
                        m_TriggerEnterIn = false;
                    }
```

```
            }
        }
    }

    /// <summary>
    /// トリガーに触れなかった灯篭
    /// </summary>
    private void DoNotReachTheMidpoint()
    {
        // 一番初めにHalfPoint2に到達した灯篭（←このif文であっていればいいなぁ…）
        if (m_TourouList.Count != 0)
        {
            m_DonotReachTourouFlag = true;
            Debug.Log(m_TourouList[0].name);
        }

        // HalfPoint2最初にたどりついたか（目的はクールタイマーを一瞬で止めないため）→
このフラグがないと一瞬しかタイマーを通らないから
        if (m_DonotReachTourouFlag)
        {
            m_EnterInCoolTime += Time.deltaTime;
            IEnumerable<Tourou> tourous = m_TourouList.Where(tourou =>
tourou.tourouDown == false && tourou.m_TriggerEnterIn == false);

            // 一番初めに到達した灯篭から3秒後
            if (m_EnterInCoolTime >= 10.0f)
            {
                if (tourous.Count(tourou => tourou.tourouDown == false &&
tourou.m_TriggerEnterIn == false) > 1)
                {
                    foreach (Tourou tourou in tourous)
                    {
                        tourou.tourouDown = true;
                    }
                }
            }
```

```
            else if (m_EnterInCoolTime <= 10.0f)
            {
                if (tourous.Count(tourou => tourou.tourouDown == false &&
tourou.m_TriggerEnterIn == false) < 0)
                {
                    m_EnterInCoolTime = 0.0f;

                    foreach (Tourou tourou in tourous)
                    {
                        tourou.tourouDown = false;
                    }
                }
            }
        }
    }

    /// <summary>
    /// 灯篭の数を確認
    /// </summary>
    private void TourouConflicted()
    {
        if (m_TriggerEnterIn)
        {
            // クールタイマー
            m_CoolTimer += Time.deltaTime;

            // HalfPoint2に進入してすぐ止めると不自然なので、進入して２秒後に灯篭を止める
            if (m_CoolTimer >= 2.0f)
            {
                m_CoolTimer = 0.0f;

                // 沈んでない灯篭をリストに追加
                foreach (GameObject tourou in
GameObject.FindGameObjectsWithTag("Tourou"))
                {
                    Tourou tourouScript = tourou.GetComponent<Tourou>();
```

```
                        if (tourouScript.tourouDown) continue;
                        m_TourouList.Add(tourouScript);
                        // 灯篭スクリプトじたいを止める
                        tourouScript.enabled = false;
                        // 灯篭のスピードを止める
                        //tourouScript.waterSpeed = 0.0f;
                        //// 灯篭を沈めさせない
                        //tourouScript.tourouDown = false;
                        //// カメラ外処理を止める
                        //tourouScript.isOutOfScreen(false);
                        // カメラの追従を止める

GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Chasing_Camera>().enab
led = false;
                        // カメラの回転フラグ
                        m_CameraIsScreen = true;
                        m_FireCharacterProduction = true;
                    }
                }
            }

            // カメラの回転フラグ
            if (m_CameraIsScreen)
            {
                // 花火の方向を向かせるためのメソッド
                TargetLookRotateAngle(m_RotateCompletion);
            }
        }

        /// <summary>
        /// ターゲット（花火）方向にカメラを回転させる
        /// </summary>
        /// <param name="completion"></param>
        /// <returns></returns>
        public bool TargetLookRotateAngle(bool completion)
        {
```

```
        // ターゲットの方向を算出
        Vector3 targetDirection = new Vector3(m_CameraTarget.position.x,
m_CameraTarget.position.y, m_CameraTarget.position.z) - Camera.main.transform.position;
        // 現在のカメラアングルからターゲット方向へ回転を得る
        Vector3 Direction = Vector3.RotateTowards(Camera.main.transform.position,
targetDirection, m_RotateSpeed, 0.0f);
        // ターゲット方向へ向かせる
        Quaternion targetAngle = Quaternion.LookRotation(Direction);

        /* RotateTowards回転 */
        // 現在のカメラアングルからターゲット方向へ回転を得る　※目的は滑らかに対象の方
向へ回転させて向かせること
        Quaternion rotate = Quaternion.RotateTowards(Camera.main.transform.rotation,
targetAngle, m_RotateSpeed * Time.deltaTime);
        Camera.main.transform.rotation = rotate;

        /* Slerp回転 */
        // 現在のカメラアングルとターゲット方向を補間する　※目的は滑らかに対象の方向へ
回転させて向かせること
        //Quaternion slerpRotate = Quaternion.Slerp(Camera.main.transform.rotation,
targetAngle, m_SlerpSpeed * Time.deltaTime);
        //Camera.main.transform.rotation = slerpRotate;

        // カメラがターゲット方向を向いているかどうか
        if (rotate == targetAngle)
        {
            completion = true;
        }
        else
        {
            completion = false;
        }

        return completion;
    }
```

```
    /// <summary>
    ///  ターゲット方向を向いていたカメラを元の方向？角度？に戻す
    /// </summary>
    /// <param name="completion"></param>
    /// <returns></returns>
    public bool MainCameraTrsnsformationRotate(bool completion)
    {
        // Unityのインスペクター上のRotation数値はオイラー角なので指定しないと0～360しか取って来ない。なので数値を変換しマイナス値も取って来れるようにする。
        float rotateX = (m_MainCameraEuler.x > 360) ? m_MainCameraEuler.x - 360 : m_MainCameraEuler.x;
        float rotateY = (m_MainCameraEuler.y > 360) ? m_MainCameraEuler.y - 360 : m_MainCameraEuler.y;
        float rotateZ = (m_MainCameraEuler.z > 360) ? m_MainCameraEuler.z - 360 : m_MainCameraEuler.z;
        //  変換したカメラの回転座標（オイラー角）をクォータニオン型にする
        Quaternion targetAngle = Quaternion.Euler(rotateX, rotateY, rotateZ);
        //  現在の角度から指定角度？座標？（元のカメラの角度）に回転させる
        Quaternion rotate = Quaternion.RotateTowards(Camera.main.transform.rotation, targetAngle, m_RotateSpeed * Time.deltaTime);
        Camera.main.transform.rotation = rotate;

        //  カメラがターゲット方向（この場合は元のカメラの角度）を向いているかどうか
        if (rotate == targetAngle)
        {
            completion = true;
        }
        else
        {
            completion = false;
        }

        return completion;
    }

    /// <summary>
```

```
    /// 火文字用のカメラと火文字を演出する
    /// </summary>
    private void FireCharacterFadeProcessing()
    {
        if (m_CameraIsScreen)
        {
            // フェードアウト
            m_FadeController.m_IsFadeOut = true;
            m_CameraIsScreen = false;
            m_CoolTimer = 0.0f;
            m_TourouEnterListFlag = false;
        }

        // フェードアウトが終了していたら
        if (m_FadeController.m_FadeOutFlag == true)
        {
            // メインカメラを一時的に非表示にする

GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Camera>().enabled =
false;
            // 火文字用カメラ
            m_FireCharacterCamera.SetActive(true);
            m_CoolTimer += Time.deltaTime;

            if (m_CoolTimer >= 2.0f)
            {
                m_CoolTimerFlag = true;
                m_FadeController.m_FadeOutFlag = false;
            }
        }

        if (m_CoolTimerFlag == true)
        {
            m_CoolTimer = 0.0f;
            m_FadeController.m_FadeOutFlag = false;
            m_FadeController.m_IsFadeOut = false;
```

```
        // フェードイン
        m_FadeController.m_IsFadeIn = true;
    }

    // フェードインが終了していたら
    if (m_FadeController.m_FadeInFlag == true)
    {
        //m_CoolTimer = 0.0f;
        //m_FadeController.m_FadeInFlag = false;
        m_CoolTimerFlag = false;

        if (m_CoolTimer >= 0.0f)
        {
            m_CoolTimer += Time.deltaTime;

            if (m_CoolTimer >= 2.0f)
            {
                m_CoolTimer = 0.0f;
                // 火文字（パーティクル）オブジェクトをアクティブ（表示？）にする
                m_FireCharacter.SetActive(true);
                m_FireProductionTimeFlag = true;
                m_FadeController.m_FadeInFlag = false;
            }
        }
    }

    if (m_FireProductionTimeFlag == true)
    {
        m_CoolTimer += Time.deltaTime;

        if (m_CoolTimer >= 10.0f)
        {
            // 火文字演出終了
            m_FireCharacterFadeFlag = true;
            m_CoolTimer = 0.0f;
```

```
                // 火文字演出が終わっていたら
                if (m_FireCharacterFadeFlag == true)
                {
                    // 花火演出フラグ
                    m_FireworksProduction = true;
                    // FireCharacterFadeProcessingメソッドを止める
                    m_FireCharacterProduction = false;
                }
            }
        }
    }

    /// <summary>
    /// 火文字用のカメラ → 花火用カメラに切り替え演出
    /// </summary>
    private void StartFireworksFadeProcessing()
    {
        // 火文字演出が終わっていたら
        if (m_FireCharacterFadeFlag == true)
        {
            // フェードアウト
            m_FadeController.m_IsFadeOut = true;
            // 火文字演出とフェードが終了
            m_FireCharacterFadeFlag = false;
        }

        // フェードアウトが終了していたら
        if (m_FadeController.m_FadeOutFlag == true)
        {
            m_FireCharacterCamera.SetActive(false);
            m_FireworksCamera.SetActive(true);
            m_CoolTimer += Time.deltaTime;

            if (m_CoolTimer >= 2.0f)
            {
```

```
                    m_CoolTimer = 0.0f;
;                   // フェードイン
                    m_FadeController.m_IsFadeIn = true;
                    m_FadeController.m_FadeOutFlag = false;
                }
            }

            // フェードインが終了していたら
            if (m_FadeController.m_FadeInFlag == true)
            {
                m_CoolTimer += Time.deltaTime;

                if (m_CoolTimer >= 3.0f)
                {
                    m_CoolTimer = 0.0f;
                    // カメラの切り替えとフェードが終了
                    m_FireworksFadeFlag = true;
                    m_FadeController.m_FadeInFlag = false;
                    m_FireworksProduction = false;

                    m_FadeController.m_IsFadeIn = false;
                    m_FadeController.m_IsFadeOut = false;
                }
            }
        }

        /// <summary>
        /// 花火終了後：花火用のカメラ → メインカメラに切り替え演出
        /// </summary>
        public void EndFirewoorksFadeProcessing()
        {
            if (m_FireworksEndingStart)
            {
                // フェードアウト
                m_FadeController.m_IsFadeOut = true;
                m_FireworksEndingStart = false;
```

```
        }

        // フェードアウトが終了していたら
        if (m_FadeController.m_FadeOutFlag == true)
        {
            // メインカメラを表示する

GameObject.FindGameObjectWithTag("MainCamera").GetComponent<Camera>().enabled =
true;
            m_FireworksCamera.SetActive(false);
            m_CoolTimer += Time.deltaTime;
            m_FadeController.m_IsFadeOut = false;

            if (m_CoolTimer >= 2.0f)
            {
                m_CoolTimer = 0.0f;
                // フェードイン
                m_FadeController.m_IsFadeIn = true;

                // フェードインが終了していたら
                if (m_FadeController.m_FadeInFlag == true)
                {
                    // 花火が終わってカメラの切り替えとフェードが終了
                    m_EndFireworksFadeFlag = true;
                    m_FireworksEndingProduction = false;
                }
            }
        }
    }
}
```