3D ゲームプログラミングⅡ／ゲームプログラミング B
第 8 回 スマートフォン用ゲーム

開発環境

Unity でスマートフォン用ゲームを開発するには、次のツールをインストールする必要がある。

Android 用

Windows、Mac とも、JDK と Android Studio が必要である。

JDK(Java SE Development Kit 8)は Oracle のサイトから、Android Studio は developer.android.com からダウンロードする。Unity の Edit ▸ Preferences を開き、External Tools の Android の Download ボタンをクリックすると、ダウンロードページが開く。

Unity、JDK、Android SDK のバージョンの組合せによっては正しく動作しないことがある。

JDK のバージョンは、Java SE 10.0.1 ではなく、Java SE 8u171 または 172 を使用する。

Android Studio をインストールすると、Android SDK がインストールされるが、tools フォルダーのバージョンを古いものに戻す必要がある。以下のリンクからダウンロード、解凍した tools フォルダーに置き換える。

http://dl-ssl.google.com/android/repository/tools_r25.2.5-windows.zip

http://dl-ssl.google.com/android/repository/tools_r25.2.5-macosx.zip

インストール後、Unity の Edit ▸ Preferences ▸ External Tools ▸ Android で、SDK と JDK のパスを入力する。SDK のパスは、Windows の場合は、/Users/[ユーザー名]/AppData/Local/Android/sdk 、Mac の場合は、/Users/[ユーザー名]/LibraryAndroid/sdk が初期値である。

iOS 用

Windows の場合、Asset Store にある iOS Project Builder for Windows（有料 $55）を使用する。

Mac の場合、Xcode が必要である。Xcode は App Store からダウンロードする。

Android スマートフォンの設定

「設定」の開発者オプションを ON にし、USB デバッグを ON にする。

開発者オプションが表示されていない場合は、「設定」の「端末情報」▸「ソフトウェア情報」で「ビルド番号」を 7 回タップする。

開発環境のテスト

Asset Store から、Simple Mobile Placeholder をダウンロード、インポートする。

Unity の Web サイトのチュートリアルの「モバイルとタッチ」にある Building を参考にして、ビルド、スマートフォンでの実行ができるかどうか確認する。

Touch 情報の取得

スマートフォンのタッチパネルからの入力情報を取得するには、次のようにする。

① Input.touchCount でタッチ数を取得する。

② タッチ数が 1 以上ならば、Input.GetTouch(index)で、Touch 構造体でタッチ情報を取得する。

③ Touch.phase の値から、タッチ状態の変化を調べる。

④ Touch.position で、スクリーン座標を取得する。

シングルタッチに限定すれば、マウスの座標と左ボタンの情報として取得できるので、PC 上のマウス入力処理と同じスクリプトで処理することも可能である。

Touch 構造体の主なメンバー

Vector2	position	スクリーン座標
TouchPhase	phase	タッチ状態
int	fingerId	指のインデクス番号（GetTouch のインデクスは信頼できない）
Vector2	deltaPosition	直前のフレームからの移動量
float	deltaTime	最後に状態が変化してからの経過時間
int	tapCount	タップ回数（Andoroid では常に 1）
float	pressure	圧力（サポートしない機種では常に 1.0）

enum TouchPhase の値

TouchPhase.Began	指が触れた瞬間
TouchPhase.Moved	指が触れたまま移動
TouchPhase.Stationary	指が触れたまま静止
TouchPhase.Ended	指が離れた
TouchPhase.Canceled	タッチ状態の追跡を中止（タッチ数の上限を超えた）

加速度センサー

Vector3 Input.acceleration を使用して、三次元の加速度ベクトルを取得できる。

ソフトウェアキーボード（モバイルキーボード）

テキスト入力が可能な領域をタップすると、自動的にソフトウェアキーボードが表示される。スクリプトから表示させる場合は、TouchScreenKeyboard.Open()を使用する。

例題 1. マウスシミュレーション

マウスドラッグのスクリプトをスマートフォンで実行する。

① 新規にプロジェクトを作成する。
② 3D Object の Sphere を作成する。
③ Sphere に次のスクリプトを追加する。

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

public class MoveScript : MonoBehaviour
{
    private Vector3 screenPoint;      // クリックされたときのスクリーン座標
    private Vector3 offset;      // ワールド座標系でのオブジェクトの位置とクリックされた位置との差

    // オブジェクト上でマウスボタンが押されたときの処理
    void OnMouseDown()
    {
        // オブジェクトの座標をワールド座標からスクリーン座標に変換
        screenPoint = Camera.main.WorldToScreenPoint(transform.position);
        // マウスのスクリーン座標
        float x = Input.mousePosition.x;
        float y = Input.mousePosition.y;
        // ワールド座標系でのオブジェクトの位置とマウスクリックされた位置との差を保存
        offset = transform.position -
                Camera.main.ScreenToWorldPoint(new Vector3(x, y, screenPoint.z));
    }

    // マウスドラッグ中の処理
    void OnMouseDrag()
    {
        // マウスのスクリーン座標
        float x = Input.mousePosition.x;
        float y = Input.mousePosition.y;
        // マウスのスクリーン座標をワールド座標に変換する
        Vector3 currentScreenPoint = new Vector3(x, y, screenPoint.z);
        //オブジェクトの位置を変更する
        transform.position = Camera.main.ScreenToWorldPoint(currentScreenPoint)
                + offset;
    }
}
```

④ シーンを保存する。

⑤ File ▸ Build Settings を選び、Build Settings Window を開く。

⑥ Add Open Scene ボタンでビルドするシーンを追加する。

⑦ Platform からスマートフォンの OS を選択し、Switch Platform ボタンをクリックする。

⑧ スマートフォンを USB ケーブルで PC に接続する。

Android の場合

⑨ Player Settings ボタンをクリックして、Inspector に PlayerSettings を表示させる。

⑩ Other Settings を展開し、Identification の Package Name を修正する。

⑪ Build And Run ボタンをクリックして、プロジェクトフォルダーに Builds フォルダーを作成し、そこに APK ファイルが作成されるように設定する。

iOS の場合

⑨ Build ボタンをクリックして、ファイル名を入力して保存する。

⑩ Xcode 用のプロジェクトフォルダーが開いたら、Unity-iPhone.xcodeproj をダブルクリックして Xcode を開く。

例題 2. スワイプ

指を滑らせることによって、オブジェクトを移動させる。

Sphere に次のスクリプトを追加する。

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

public class TouchScript : MonoBehaviour
{
    private int currentIndex = -1;   // スワイプ中の指 ID
    private Vector3 screenPoint;      // タップされたときのスクリーン座標
    private Vector3 offset;    // ワールド座標系でのオブジェクトの位置とタップされた位置との差

    void Update ()
    {
        // タッチがある場合
        if (Input.touchCount > 0)
        {
            for (int i = 0; i < Input.touchCount; i++)
            {
                Touch touch = Input.GetTouch(i);
                switch (touch.phase)
                {
                    // タップされた瞬間
```

```
case TouchPhase.Began:
    // スワイプ中のものがない場合
    if (currentIndex < 0)
    {
        // オブジェクト上でタップされたかどうか調べる
        Ray ray = Camera.main.ScreenPointToRay(touch.position);
        RaycastHit hit = new RaycastHit();
        if (Physics.Raycast(ray, out hit))
        {
            if (hit.collider.gameObject == gameObject)
            {
                // オブジェクトの座標をワールド座標から
                // スクリーン座標に変換
                screenPoint = Camera.main.WorldToScreenPoint(
                    transform.position);
                // タップのスクリーン座標
                float x = touch.position.x;
                float y = touch.position.y;
                // ワールド座標系でのオブジェクトの位置と
                // タップされた位置との差を保存
                offset = transform.position -
                    Camera.main.ScreenToWorldPoint(
                    new Vector3(x, y, screenPoint.z));
                // 指 ID の保存
                currentIndex = touch.fingerId;
                // 色を赤くする
                GetComponent<Renderer>().material.color =
                    Color.red;
            }
        }
    }
    break;
// スワイプ中
case TouchPhase.Moved:
    // 指 ID が一致する場合
    if (currentIndex == touch.fingerId)
    {
        // スワイプのスクリーン座標
        float x = touch.position.x;
        float y = touch.position.y;
        // スワイプのスクリーン座標をワールド座標に変換する
        Vector3 currentScreenPoint =
            new Vector3(x, y, screenPoint.z);
        //オブジェクトの位置を変更する
        transform.position = Camera.main.ScreenToWorldPoint(
            currentScreenPoint) + offset;
    }
    break;
// 指が離された瞬間
case TouchPhase.Ended:
```

```
                        // 指 ID が一致する場合
                        if (currentIndex == touch.fingerId)
                        {
                            // 指 ID をリセット
                            currentIndex = -1;
                            // 色を白くする
                            GetComponent<Renderer>().material.color = Color.white;
                        }
                        break;
                    case TouchPhase.Canceled:
                        if (currentIndex == touch.fingerId)
                        {
                            currentIndex = -1;
                            GetComponent<Renderer>().material.color = Color.white;
                        }
                        break;
                }
            }
        }
    }
}
```

例題 3. ピンチ

二本の指の間隔を広げることによってズームする。

次のスクリプトをカメラに追加する。

```
using UnityEngine;

public class PinchZoom : MonoBehaviour
{
    public float perspectiveZoomSpeed = 0.5f;    // 透視投影モードでの有効視野の変化の速さ
    public float orthoZoomSpeed = 0.5f;    // 平行投影モードでの平行投影サイズの変化の速さ


    void Update()
    {
        // 端末に 2 つのタッチがある場合.
        if (Input.touchCount == 2)
        {
            // 両方のタッチを格納する
            Touch touchZero = Input.GetTouch(0);
            Touch touchOne = Input.GetTouch(1);

            // 各タッチの前フレームでの位置を求める
            Vector2 touchZeroPrevPos = touchZero.position - touchZero.deltaPosition;
            Vector2 touchOnePrevPos = touchOne.position - touchOne.deltaPosition;
```

```
                // 各フレームのタッチ間のベクター（距離）の大きさを求める
                float prevTouchDeltaMag =
                        (touchZeroPrevPos - touchOnePrevPos).magnitude;
                float touchDeltaMag = (touchZero.position - touchOne.position).magnitude;

                // 各フレーム間の距離の差を求める
                float deltaMagnitudeDiff = prevTouchDeltaMag - touchDeltaMag;

                // カメラが平行投影の場合
                if (Camera.main.orthographic)
                {
                    // タッチ間の距離の変化に基づいて平行投影サイズを変更する
                    Camera.main.orthographicSize +=
                            deltaMagnitudeDiff * orthoZoomSpeed;

                    // 平行投影サイズが 0 未満にならないようにする
                    Camera.main.orthographicSize = Mathf.Max(
                            Camera.main.orthographicSize, 0.1f);
                }
                // カメラが透視投影の場合
                else
                {
                    // そうでない場合は、タッチ間の距離の変化に基づいて有効視野を変更する
                    Camera.main.fieldOfView +=
                            deltaMagnitudeDiff * perspectiveZoomSpeed;

                    // 有効視野を 0 から 180 の間に固定する
                    Camera.main.fieldOfView =
                            Mathf.Clamp(Camera.main.fieldOfView, 0.1f, 179.9f);
                }
            }
        }
}
```

例題 4. 加速度センサー

スマートフォンを傾けることよって、オブジェクトを移動させる。

① Main Camera の Position を(0, 10, 0)に、Rotation を(90, 0, 0)に変更する。

② Sphere に次のスクリプトを追加する。

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

public class AccelScript : MonoBehaviour
{
```

```
    public float speed = 1f;       // 移動速度

    void Update()
    {
        // 2 方向の加速度からベクトル作成
        Vector3 dir = new Vector3(Input.acceleration.x, 0, Input.acceleration.y);
        // ベクトルの長さが 1 を超えないようにする
        if (dir.sqrMagnitude > 1)
            dir.Normalize();
        // オブジェクトを動かす
        transform.Translate(dir * speed * Time.deltaTime);
    }
}
```