3.2 Unity（続き）

　続いて、Unityで行える各種設定について簡単に説明します。

■Sceneビュー表示位置を切り替える（オブジェクト操作時の視点変更

視点移動ツール（手のひらのアイコンで表示されています）で切り替えができます。

そのままSceneビューの画面をドラッグすることで、平行に視点切り替えを行えます。

Altキーを押しながらSceneビューの画面をドラッグすることで、回転ができます。

■Gameビューの表示位置を切り替える（カメラ移動によるプレイヤー視点の変更）

Hierarchyビューの「Main Camera」を移動させることで、ゲーム時の視点を変更することができます。

「Main Camera」をクリックし、平行移動ツール・回転ツール・スケールツールなどでカメラの位置と向きを変えることができます。

■テクスチャーマッピング（Texture Mapping）

オブジェクトの表面にテクスチャー（模様）を張り付けることができます。

Windowsのピクチャフォルダなどから画像ファイルをProjectビューのAssetsフォルダにドラッグ＆ドロップさせることで、今作成しているプロジェクトに追加できます。

（沢山のテクスチャーを使う場合は、Assetsフォルダのサブフォルダとして「Texture」などの名称で新規作成してそこにドロップしたほうが、管理が楽になるでしょう）

そして、追加したテクスチャーをHierarchyビューのオブジェクトにドラッグ＆ドロップするだけです。テクスチャーのある階層の下にMaterialsというフォルダが自動作成されます。

■プレハブ（Prefab）

オブジェクトの元となる雛型を作成しておくことで、同じ形状や特徴を持ったオブジェクトを簡単に複数作成することができるようになります。この雛型をプレハブと呼びます。

Projectビューから「Create」→「Prefab」を選んでプレハブを作成します。（デフォルトの名前は「New Prefab」です。）

そして、プレハブ化したいオブジェクトをHierarchyビューからいま作成されたプレハブにドラッグ＆ドロップします。アイコンが変わったことを確認してください。

以降、出来上がったプレハブをHierarchyビューにドロップすることで、この雛型に沿ったオブジェクトをいくつも作ることができます。これをインスタンス化と呼びます。

プレハブに加えた変更は、そのプレハブからインスタンス化されたすべてのオブジェクトに反映されます。

■オブジェクトの階層化

いくつものオブジェクトをまとめてひとつのオブジェクトとして取り扱う事ができます。そのために行うことが階層化です。

GameObjectメニューから「Create Empty」を選びことで、Hierarchyビューに「GameObject」というオブジェクトができます。これが、形を持たない空（から）オブジェクトです。

続いて、ひとつにまとめたいオブジェクトをこの空オブジェクトにドラッグ＆ドロップします。空オブジェクトの下位層にドロップしたオブジェクトが追加されていきます。

以降、上位となる空オブジェクトを移動や回転などさせると、すべての下位オブジェクトがひと固まりとなって移動や回転されます。

■タグ（Tag）

オブジェクトの状態や種類を示すためのタグというものを定義することができます。

たとえば「敵キャラクターのオブジェクトすべてに同じタグ名をつけておき、そのタグが付いたオブジェクトに接触（衝突）したときにプレイヤーキャラクターがダウンする」などといった使い方ができます。

Editメニューから「Project Settings」→「Tags」と選ぶとInspectorビューにTagManagerが表示されます。その中にあるTagsを開いて、Element0の右横にタグ名を入力します。（Element0へ入力されると自動的にElement1が追加されますので、いくつもタグ名を定義することができます）

Playerなど、あらかじめいくつかのタグは定義されていますのでそれを使うこともできます。

タグのオブジェクトへの割り当て（タグ付け）は、Hierarchyビューでオブジェクトを選んでInspectorビューの上部にある「Tag」というドロップボックスで行います。

■トリガー

オブジェクトをトリガーという状態にすることで、他のオブジェクトが衝突してきたときに検知を行って対応する処理を行わせることができます。

プレイヤーキャラクターと敵キャラクターの衝突判定や、ゴール地点や得点取得できる地点・罠の位置などにプレイヤーキャラクターが入ってきたことの判定などを行う事ができるようになります。

尚、トリガー化されたオブジェクトに他のオブジェクトが衝突してきても、反射せずにすり抜けるようになります。

オブジェクトをトリガー化するには、Hierarchyビューでオブジェクトを選んだあとInspectorビューにある「Is Trigger」のチェックボックスにチェックを入れます。

トリガーとなったオブジェクトに他のオブジェクトが衝突（進入）してきたことは、そのトリガーに割り当てられたスクリプトで判断して処理することができます。

|  |  |  |
| --- | --- | --- |
| **【演習：Unity その2】**  今回のプリントで説明している機能を使ったアプリケーションを作ってみましょう。  所々でPlayして動作確認をしてみながら作成を楽しんでみてください。  1. 新しくプロジェクトを作成する  (1)Unityを起動する  (2)Fileメニューから「New Project...」を選ぶ  (3)任意の「Project Location」を指定して「Create」ボタンを押す  （「New Unity Project」等のフォルダ名をわかり易いものに変えましょう）  2. 基本となるオブジェクトを作成する  (1)Hierarchyビューの「Create」から「Directional light」を選ぶ  (2)同じく「Create」から「Sphere」、「Cube」、「Cylinder」を選ぶ  　（Cylinderは円柱を表すオブジェクトです）  (3)Hierarchyビュー上に表示されている「Cube」で右クリックして「rename」を選ぶ  (4)「Cube」の名前を「TargetCube」に変える  (5)同様に以下の通り名前を変える  　　「Cylinder」 →「MovePin1」  　　「Sphere」 →「PlayerBall」  (6)Hierarchyビューから「TargetCube」を選ぶ  (7)Inspectorビューの「Transform」欄にある「Position」「Rotation」「Scale」のそれぞれの(X,Y,Z)の値を順に、(70, -40,10) (0, 0, 0) (10, 2, 80) に変える  (8)同様に「PlayerBall」の値を (0, 50,20) (0, 0, 0) (5, 5, 5) に変える  (9)同様に「MovePin1」の値を (0, 30,-15) (90, 0, 0) (10, 30, 10) に変える  (10)同様に「Main Camera」の値を (0, 20,-100) (0, 0, 0) (1, 1, 1) に変える  3. テクスチャーをオブジェクトに貼りつける  (1)好きな画像をProjectビューの「Assets」欄にドラッグ＆ドロップ（D&D）する  (2)「Assets」欄の画像（テクスチャー）をHierarchyビューのオブジェクトにD&Dする  4. プレハブを作成する  (1)Projectビューの「Create」から「Prefab」を選ぶ  (2)「New Prefab」の名前を「MovePin」に変える  (3)Hierarchyビューの「MovePin1」を「MovePin」にD&Dする  5. プレハブを使ってインスタンスオブジェクトを作成する  (1)Projectビューの「MovePin」（プレハブ）をHierarchyビューの何も表示されていない場所にD&Dする  (2)Hierarchyビューに作成された「MovePin」の名前を「MovePin2」に変える  (3)同様に「MovePin3」～「MovePin6」までを作成する  6. 空オブジェクトを使用して構造化されたオブジェクトを作成する  (1)GameObjectメニューから「Create Empty」を選ぶ  (2)作成された「GameObject」の名前を「MovePins」に変える  (3)Hierarchyビューの「MovePin1」～「MovePin6」を「MovePins」にD&Dする  (4)「MovePins」を開いて下位層として「MovePin1」～「MovePin6」がある事を確認する  (5)「MovePin1」の「Transform」欄の「Position」の値を (0, 30, -15)に変える  (6)同様に以下の通り「Position」の値を変える  　　「MovePin2」　→　( -20, 10, -15)  　　「MovePin3」　→　( 20, 10, -15)  　　「MovePin4」　→　( -40, 0, -15)  　　「MovePin5」　→　( 0, 0, -15)  　　「MovePin6」　→　( 40, 0, -15)  7. オブジェクトに物理属性を与える  (1)Hierarchyビューで「PlayerBall」を選ぶ  (2)Inspectorビューの「Add Component」ボタンを押し「Physics」「Rigidbody」を選ぶ  (3)同様に「MovePins」にも「Rigidbody」を加える  (4)「MovePins」の「Rigidbody」欄の「Is Kinematic」にチェックを入れる  (5)Projectビューの「Create」から「Physic Material」を選ぶ  (6)作成された「New Physic Material」の名前を「Reflect」に変える  (7)「Reflect」をクリックしてInspectorビューの「Bounciness」の値を0.8に変える  (8)「Reflect」をHierarchyビューの「PlayerBall」と「MovePins」にD&Dする  9.スクリプト（カーソルキーによる操作）を作成する  (1)ProjectビューのCreateメニューから「JavaScript」を選ぶ  (2)作成された「NewBehaviourScript」の名前を「MoveScript」に変更する  (3)「MoveScript」をダブルクリックしてスクリプトエディタ（MonoDevelop）を起動する  (4) 次のスクリプトを入力して保存する   |  | | --- | | function Update () { // 一定時間ごとに行われる処理  transform.position.x +=  Input.GetAxis("Horizontal") \* (30.0 \* Time.deltaTime);  } |   (5)「MoveScript」をHierarchyビューの「movePins」にD&Dする  8. オブジェクトをトリガーに変える  (1)Hierarchyビューで「TargetCube」を選ぶ  (2)Inspectorビューの「Box Colider」欄の「Is Trigger」にチェックを入れる  9. オブジェクトにタグを付ける  (1)Editメニューから「Project Settings」→「Tags」を選ぶ  (2)「Tags」を開いて「Element 0」の右にPlayerBallTagと入力する  (3)HierarchyビューのPlayerBallを選ぶ  (4)Inspectorビューの上部にあるTagを「PlayerBallTag」に変える  9. スクリプト（トリガーの処理）を作成する  (1)ProjectビューのCreateメニューから「JavaScript」を選ぶ  (2)作成された「NewBehaviourScript」の名前を「TriggerScript」に変更する  (3)「TriggerScript」をダブルクリックしてスクリプトエディタを起動する  (4)次のスクリプトを入力して保存する   |  | | --- | | private var goalFlag : boolean;  function Start () { // 実行前に行われる  goalFlag = false;  }  function OnTriggerEnter (other : Collider) { // トリガー進入時に行われる  if (other.gameObject.tag.Equals("PlayerBallTag")) {  goalFlag = true;  }  }  function OnTriggerExit (other : Collider) { // トリガー退場時に行われる  }  function OnGUI () { // GUI処理を記述する特別なメソッド  if (goalFlag == true) {  GUI.Label(Rect(10, 10, Screen.width - 20, Screen.height - 20), "Goal!");  }  } |   (5)「TriggerScript」を「TargetCube」にD&Dする  10. シーンとプロジェクトを保存する  (1)Fileメニューから「Save Scene」を選ぶ  (2)ファイル名を指定して保存する  (3)Fileメニューから「Save Project」を選ぶ  11. 動作確認する  　カーソルキーの左右で、ピン（シリンダー）群が左右に移動します。  　ピンでうまく跳ねさせながら、左下の板にボールをぶつけてください。  　板にボールが当たれば、画面の左上に「Goal!」と表示されます。 |

応用練習

　・以下のスクリプトを「PlayerBall」に割り当てるとマウスクリックで開始になります。

|  |
| --- |
| function Start () { // 開始時は重力影響なし  this.rigidbody.useGravity = false;  }  function Update () { // マウスクリックで重力開始  if (Input.GetButtonDown("Fire1")) {  this.rigidbody.useGravity = true;  }  } |

　・Webで調べてスクリプト等の勉強をしましょう（Input、GUIなどの活用方法等）

|  |
| --- |
| **課題2（Unityの作品をGitHubで提出）**  　・Unityで、先生が見て楽しめるような作品を作りGitHubに公開(提出)してください  　※※※　注意！　※※※  　　・プロジェクトのフォルダ名に空白を含んでいるとGitHubで不具合が出ます  　　・Unity使用中はコミットできません（Unityがロックするファイルがある為）  **提出方法：担当の先生にメールで、課題のURL（GitHubリポジトリのページ）を伝える** |