**○Unity Hubの使い方**

・プロジェクト名は最初にのみ決定できて、後から変更することができない

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

| **名前** | **説明** |
| --- | --- |
| Scene(シーン) | シーンにあるGameObjectを実際に動かしたり操作できる画面 |
| Game(ゲーム) | ゲームで表示される画面 |
| Hierarchy | シーンの中に表示されるGameObjectのリスト |
| Project | プログラムや画像など、ファイルのリスト |
| inspector | GameObjectのコンポーネントやプロパティを設定する場所 |

・セーブは「上のタブのFile」 → Save

・画面のレイアウトをデフォルトにするときは「画面右上のLayout」→ Default

・カメラはゲームの映像

・ライトは明かり

**○オブジェクト**

・オブジェクトを設置するときには「上のタブのGameObject」から「3D Object」を選択するか「Hierarchyウィンドウから右クリック」で「3D Object」を選択する

・3D Object一覧

|  |  |
| --- | --- |
| Cube | 直方体 |
| Sphere | 球 |
| Capsule | カプセル |
| Cylinder | 円柱 |

・場所、向き、大きさをInspectorウィンドウかマウスで指定できる

・マウスで変更するときには最初に操作パネルで何を変更するかをクリックしてから矢印をドラッグする。十字矢印で場所、円矢印で向き、正方形矢印で大きさ

ダイアグラム, ホワイトボード

自動的に生成された説明

・マウスホイールを回して視点をズームイン、ズームアウトができる。

グラフ が含まれている画像

自動的に生成された説明

・マウスホイールをドラッグすると視点の変更ができる。右クリックドラッグで視点の角度を変更できる

人, スポーツゲーム, シャツ が含まれている画像

自動的に生成された説明図形 が含まれている画像

自動的に生成された説明

**○カメラ**

・HierarchyウィンドウからMain Cameraを選択するとカメラが映す範囲が確認できる

・Main Cameraで映している内容をSceneビューに表示したいときは「HierarchyからMain Cameraをクリック」→「GameObject」→「Align view to selected」とする

・「Align view to selected」は選択しているオブジェクトからの視点にするということ

**○Gameビュー**

・Gameビューでカメラの映像が確認できる

・▷が再生ボタン、エラーがなければ実行される

・再生中に変更した内容は反映されずに消えてしまうので注意！必ず再生を終了してから変更すること！

・再生中は▷が青くなる

**○Material**

・下にあるProjectビューで右クリックをして「Create」→「Material」

・Projectビューに作ったMaterialをHierarchyビューのオブジェクトにドラッグすると反映される。または、オブジェクトの「Mesh Renderer」の「Element 0」の◎を選択してから反映させたいMaterialをクリックしてもできる

・Albedoは色

・Metallicは金属光沢

・Smoothnessは表面のつや

・Emissionは発光度合い、物体自体が光源となる

・写真をオブジェクトに貼り付けたいときは写真をドラッグ&ドロップするとProjectビューに入るので、その写真をオブフェクトにドラッグするか、反映させてあるMaterialのAlbedoの◎をクリックして画像を選択する

**○Asset**

・Assetとはゲームの素材や部品のこと。オブジェクトに質感を持たせたり、模様をつけたりすることができる。[Unity Asset Atore](https://assetstore.unity.com/)でダウンロードしたら使えるようになる

・「Add to my assets」→「Accept」→「Open in Unity」→「Unity Editorで開く」→「Package Managerからダウンロード」→「Import」→「Import」をするとProjectビューの中に保存されてあるはず

例、Unity chanからUnity chan tileをCubeに反映させる

Projectビューにある「Asset」→「unity-chan!」→「Unity-chan! Model」→「Art」→「Stage」→「Materials」にある「Unity chan tile」をHierarchyビューにあるCubeにドラッグする

例、Unity chanからキャラクターであるUnity chanを画面に出す

Projectビューにある「Asset」→「unity-chan!」→「Unity-chan! Model」→「Prefabs」→「for Locomotion」→「unitychan」をSceneビューにドラッグする

・今までダウンロードしたAssetを使いたいときは「Window」→「Package Manager」→「My assets」からAssetの名前をクリックしてImportする

・既に置いてあるオブジェクトと同じ位置にModelを置きたいときは、Modelをオブジェクトにドラッグしてオブジェクトの下の階層に入れる。元のオブジェクトの「Mesh Renderer」からチェックを外して見えなくしてから、Modelの大きさを変更するとよい

**○RigidBody（物理法則）**

・RigidBodyはオブジェクトに物理法則を付与するComponent（部品）

・RigidBodyを付与したいオブジェクトのInspectorビューの一番下にある「AddComponent」から「RigidBody」を検索して選択する

| **名前** | **説明** |
| --- | --- |
| Mass | 物体の質量 (Kg 単位) |
| Drag | 力によって動く際に、オブジェクトに影響する空気抵抗の大きさ。 |
| Angular Drag | トルク(軸を回す力)によって回転する際に、オブジェクトに影響する空気抵抗の大きさ |
| Use Gravity | チェックを入れると重力の影響を受ける |
| Is Kinematic | チェックを入れると物理演算が無効になる |
| Freeze Position | 選択した軸でRigidbodyによる移動を停止する |
| Freeze Rotation | 選択した軸でRigidbodyによる回転を停止する |

・Rigidbody ConstraintsでFreeze Positionで動きを止めて、Freeze Rotationを1方向だけ残すと移動せずに回転だけするオブジェクトができる！

・動いている物体の動きを止めるときには速度を0にすればよい。

rb.velocity = new Vector(0f, 0f, 0f);

**◯角度の表し方**

【オイラー角】

・オイラー角とはオブジェクトの角度をX、Y、Zの3つの軸で表す方法

・下の写真のRotation

メーター が含まれている画像

自動的に生成された説明

・スクリプトで変更するには「transform.Rotate(x, y, z);」とする

【クオータニオン(Quaternion)】

・クオータニオンとは自分の好きな方向に軸を取り、その軸に回転を与えることでオブジェクトを回転させる方法

・オイラー角とは違い、回転軸(ベクトル)の場所を示す3つの数字と、回転角(回転の大きさを表す)数字の4つで表される

**〇C#コードの意味**

・オブジェクトに細かい指示を与えるプログラムを付与することができる。オブジェクトのInspectorウィンドウの一番下にある「Add Component」をクリックして一番下にある「New Script」からファイル名を決めて生成する

・VS Codeのファイル名の右に「\*」が着いていたら保存していない状態。保存したら「\*」が消える。保存は「Ctrl + S」か「ファイルのマークをクリック」

・最初からStart関数とUpdate関数が作られている

・void Start()関数は初期状態の設定

・void Update()関数は状態を変更するときに用いる関数。キーボード操作をしたときにジャンプしたり移動させたりするのはこの中で行う

・代表的な関数

| **名前** | **説明** |
| --- | --- |
| Debug.Log(値) | consoleに引数の値を表示 |
| Random.Range(min, max) | ランダムでminとmaxの間の値を返す |
| transform.LookAt(Transform) | 指定された座標の向きにアクセス |
| transform.Translate(x, y, z) | 入力されたx, y, zだけ移動 |
| Application.Quit() | アプリケーションを終了 |

例、x軸方向に等速直線運動させる

void Update(){ transform.Translate(0.1f, 0, 0); }

// 1/60秒ごとに0.1f進む→1秒で6f進む

例、すり抜けたときにOnTriggerEnterを使う

オブジェクトのInspectorウィンドウで「Collider」→「Is Triggerにチェックを入れる」と物理接触をなくすことができるので他のオブジェクトをすり抜けるようになる

・transform.positonで座標, transform.Rotationで向きにアクセスすることができる

・x軸, y軸, z軸それぞれの方向の単位ベクトルがほしいときはtransform.right, up, forwardを用いる。

例、ジャンプ

rb.AddForce(transform.up \* jumpPower); // これは(0f, jumpPower, 0f)と同じ

例、横方向に２移動させる（単位ベクトルなので積をとることに注意）

transform.position = transform.position + transform.right \* 2;

・「using UnityEngine.UI」を最初に宣言するとテキストを使うことができるようになる

・publicは違うファイルやInspectorビューからもアクセス可能であり、privateはクラス内部でのみアクセス可能である

・Rigidbodyを適応すると物理演算が可能になる。Scriptでは宣言で「private Rigidbody rb;」としてStart()関数の中で初期状態を「rb = GetComponent<Rigidbody>();」とする

・キーボードを押したときの挙動は「Input.GetKeyDown(“キーの名前”)」としてUpdate関数の中に入れると、指定したキーが押されたときにtrueとなるのでif文で表現する

・ジャンプするときは上向きの力を加える「rb.AddForce(transform.up \* jump\_power, ForceMode.Force);」

・特定のオブジェクトの向きを追うには「transform.lookAt(Transform)」とする。引数には追う対象のオブジェクトのTransformを入れる

例、相手のゴールの向きに合わせてから力を前方に加える

public Transform goal;

void Update(){

transform.LookAt(goal);

packrb.AddForce(transform.forward \* 10f, ForceMode.Impulse);

}

transform.forwardを用いるときは向きに注意！逆のときはマイナスをかけるのを忘れずに

**〇イベント関数**

・イベント関数とは普通の関数とは異なり、特定のタイミングのイベント発生時に呼ばれるようにUnity側で予約されている関数のこと。イベント関数と同じ名前の関数を作るとバグが発生するので注意

・代表的なイベント関数

| **名前** | **説明** |
| --- | --- |
| void Start(){ } | ゲームが始まると最初の１回だけ呼ばれる |
| void Update(){ } | 1秒間に60回ごとに繰り返し呼ばれる |
| void FixedUpdate(){ } | 指定した時間ごとに繰り返し呼ばれる |
| void OnDestroy(){ } | オブジェクトが破壊されたとき呼ばれる |
| void OnCollisionEnter(Collision collision){ } | オブジェクト同士が衝突したときに呼ばれる |
| void OnCollisionExit(Collision collision){ } | オブジェクト同士が離れたときに呼ばれる |
| void OnTriggerEnter(Collider other){ } | 物体がすり抜けたときに呼ばれる |
| void OnMouseDown(){ } | マウスが押されたときに呼ばれる |

・OnCollisionEnterで引数として用いるCollisionはunityから以下の様々な情報を取得できる

| **名前** | **説明** |
| --- | --- |
| gameObject | 相手のGameObjectの情報(読み取り専用) |
| rigidbody | 相手のRigidbodyの情報(読み取り専用)相手に対象がない場合はnull |
| transform | 相手のTransformの情報 |
| articulationBody | 相手のArticulationBodyの情報(読み取り専用) |
| body | 相手のRigidbodyもしくはArticulationBodyの情報(読み取り専用) |
| collider | 相手のColliderの情報(読み取り専用) |
| contactCount | 相手のcollision数 |
| impulse | 相手と自分の合計衝撃値 |
| relativeVelocity | 相手と自分の相対速度(読み取り専用) |

・取得するときは「collision.名前」とする

例、ぶつかった相手の名前、座標、大きさ、回転を表示する

**void** **OnCollisionEnter**(Collision collision)

{

Debug.Log(collision.gameObject.name);

Debug.Log(collision.transform.position);

Debug.Log(collision.transform.localScale); // transform.scaleではない

Debug.Log(collision.transform.rotate); // unityの表示と異なる記法の角度

}

・ぶつかったObjectの座標の変更をするときにはnew Vector3(x, y, z)とする

collision.transform.position = new Vector3(0, 10, 0);

・マウスイベント関数

| **名前** | **説明** |
| --- | --- |
| void OnMouseDown(){ } | マウスボタンが押された時 |
| void OnMouseDrag(){ } | マウスボタンが押された状態でマウスを移動させている間 |
| void OnMouseUp(){ } | マウスボタンを離した時 |
| void OnMouseEnter(){ } | マウスが乗った時 |
| void OnMouseExit(){} | マウスが離れた時 |
| void OnMouseOver(){ } | マウスが乗っている時 |

**◯EventTrigger関数**

・EventTrigger関数ではオブジェクトに対してのマウスのクリックやドラッグを取得する

・EventTriggerの使い方

オブジェクトを動かす自作関数MoveCube関数をクリックした際に作動したい

1. 使いたいオブジェクトにAddComponentからEvent Triggerをつけ、Event Triggerの中のAdd New Event Typeからタイプを選択する。右下のプラスボタンから動かしたいObjectをドラッグアンドドロップする。No functionからスクリプト - MoveCube()を選択
2. オブジェクトに当たり判定Colliderがついているかどうか確認
3. EventSystemをシーンに追加する。左クリック - UI - Event System
4. カメラにAddComponentから「PhysicsRaycaster」をつける。

・代表的なEventTrigger関数

| **名前** | **説明** |
| --- | --- |
| OnPointerEnter | ポインターがオブジェクトに乗ったときに呼び出されます |
| OnPointerExit | ポインターがオブジェクトから離れたときに呼び出されます |
| OnPointerDown | ポインターがオブジェクトを押下したときに呼び出されます |
| OnPointerClick | あるオブジェクト上でポインターを押し、同一のオブジェクト上で離した際に呼び出されます |

**〇C#の基本**

・int型とString型を+演算子で合わせて、int型をString型に読み替えることができる

int score = 0;

String text;

void Update(){ text.text = “Score : ” + score; } // Score : 0

**〇よくやること**

【キーボード操作】

・タイミング

| **項目** | **説明** |
| --- | --- |
| GetKey | キーを押している間 |
| GetKeyDown | キーを押した時 |
| GetKeyUp | キーを離した時 |

・キーボードが押されたときにtrueを返す関数とif文を用いる

| **項目** | **説明** |
| --- | --- |
| KeyCode.A | Aキー |
| KeyCode.Space | スペースキー |
| KeyCode.Return | エンターキー |
| KeyCode.LeftArrow | 左矢印 |

[他のキーについて](https://docs.unity3d.com/ja/2022.1/ScriptReference/KeyCode.html)

例、スペースボタンを押したとき

if(Input.GetKey(KeyCode.Space)) Debug.Log(“スペースキーが押されている”);

・左矢印を押した時間（-1が最高）、右矢印を押した時間（+1が最高のように-1から1までの浮動小数点を返す関数「Input.GetAxisRaw(“Horizontal”)」”Horizontal”を”Velocity”に変えたら上下版になる。

float direction = Input.GetAxisRaw(“Horizontal”);

【オブジェクトの移動】

・オブジェクトを動かしたいとき

①transform.Translate(x, y, z)をして瞬間移動させる

②AddForceを使って力を加える

③velocityを変えて速度を変化させる、といった3つの方法がある。

・RigidBodyの値をいじるときにはクラスの先頭で「public Rigidbody rb;」として「rb.関数名」で使えるようにする。初期化はStart関数の中で「rb = GetComponent<Rigidbody>();」とすること。

・RigidBodyの値を変更する際にはUnityのrbにオブジェクトをアタッチする（けっこう忘れやすいので注意！）

・力を加えるときは「rb.AddForce(new Vector3(x, y, z));」とする。

・速度を変えるときは「rb.velocity = new Vector3(x, y, z);」

・ジャンプするときは上向きの力を加える「rb.AddForce(transform.up \* jump\_power, ForceMode.Force);」ジャンプ中にもう一度ジャンプしないようにするためにジャンプしたときにtrueになり、床にぶつかったときにfalseにするフラグを用いるとよい。

【AddForceについて】

・ForceModeで力の設定をする。AddForce関数の第二引数にする。確証はないが基本的にForceMode.Forceで良いかも？？

| **変数** | **意味** |
| --- | --- |
| ForceMode.Force | 物体の質量を利用して、物体に継続的な力を加える |
| ForceMode.Acceleration | 物体の質量を無視して、物体へ継続的な加速を加える |
| ForceMode.Impulse | 物体の質量を利用して、物体に瞬間的に力を加える |
| ForceMode.VelocityChange | 物体の質量を無視して、物体に瞬間的に速度変化を起こす |

例、ジャンプ

private Rigidbody rb;

float jumpPower = 300f;

if(Input.GetKeyDown(KeyCode.Space)){

rb.AddForce(transform.up \* jumpPower, ForceMode.Force);

}

【上下左右の移動】

・上下左右のキー入力を取得するには「Input.GetAxisRaw(“Horizontal or Vertical”)」として、transform.Translateをもちいて変更する。返ってくるのは0から1の間の小数なので、後で大きさをかけてあげると良い。

float speed = 0.01f;

float direction\_h = Input.GetAxisRaw("Horizontal");

float direction\_v = Input.GetAxisRaw("Vertical");

transform.Translate(direction\_h \* speed, direction\_v \* speed, 0);

・Input.GetAxisでもよい。軸の補完がされるらしい？

・マウスの移動量も同じようにして取得することができる。Input.GetAxis(“Mouse X”)でx軸方向、Input.GetAxis(“Mouse Y”)でy軸方向の移動量を示す。

void OnMouseDrag(){

mouse\_x = Input.GetAxis("Mouse X");

mouse\_y = Input.GetAxis("Mouse Y");

transform.Translate(mouse\_x \* speed, 0f, mouse\_y \* speed);

}

【力のため方】

・キーボードを押している最中に力をためて、離したときに開放する

・Input.GetKeyで押している最中に力をため続けて、Input.GetKeyUpで離したときに開放する

・Sliderで表示するのを考慮して0から1を行き来すると良い

例、スペースキーを押す長さでパワーを調節

// Spaceを押している最中に力を貯める

if(Input.GetKey(KeyCode.Space)){

power += 0.01f;

slider.value = power;

if(power > 1f) power = 0f;

}

// Spaceを離したら発射

if(Input.GetKeyUp(KeyCode.Space)){

rb.AddForce(obj.transform.forward \* power \* 1000, ForceMode.Force);

power = 0f;

}

【摩擦】

・摩擦を付けるにはAssetで左クリック – Physics Mateerialを選択して、目的のオブジェクトについているColliderのMaterialにドラッグして入れる

【Destroy】

・自分自身のObjectを消すときには「Destroy(gameObject, 時間);」とする。gは小文字なことに注意！時間は指定した時間が経過したら消すことができる。時間の引数は省略可能。また、衝突した相手を特定したいときには「other.gameObject.name」で確認する。

例、衝突したときに自分を0.5秒後に消える

void OnCollisionEnter(Collision other){

if(other.gameObject.name == “Ball”) Destroy(gameObject, 0.5f);

}

・衝突した相手(OnCollisionEnter)を消すときは「Destroy(collision.gameObject);」とする

・通過した相手(OnTriggerEnter)を消すときには「Destroy(other.gameObject);」とする

・任意のオブジェクトを消す際には先頭で「public GameObject 名前;」と宣言してから「Destroy(名前);」とすればよい

**〇UI（テキストなど）**

・ゲームビューに文字を表示するにはUIと呼ばれるゲームビューに文字やボタンを表示するためのオブジェクトを使用する

・UIはCanvasと呼ばれるUI専用の枠の中に作成される

| **UIの種類** | **機能** |
| --- | --- |
| Text | 文字を表示する |
| Image | 画像を表示する(Spriteテクスチャのみ) |
| Panel | 複数のUIをまとめて表示/非表示にする |
| Slider | 数の大きさを示すバーを表示する |
| Button | クリックできるボタンを表示する |
| Input Field | 文字を入力できる枠を表示する |
| Canvas | UIを配置することができる領域 |

【Text】

・「Game Object」→「UI」→「Legacy」→「Text」で生成される

・Sceneビューではかなり手前に生成されるので画面を引かないと見えない。Gameビューにすれば見えるはず

・画面で表示される位置を設定するときは画面を引いて矢印を動かす方が楽かも

・基本的に「Horizontal」も「Vertical」も「Overflow」にしておくと良い

・Textの設定

| **名前** | **説明** |
| --- | --- |
| Font | テキストを表示するために使用するフォント |
| Font Style | テキストに適用するスタイル |
| Font Size | 表示するテキストのサイズ |
| Line Spacing | テキストの縦方向の行間 |
| Aliginment | テキストの縦方向の行間 |
| Overflow | 枠からはみ出しても文字を表示するか |
| Bestfit | 自動で枠内に文字を収めるか |
| Color | テキストの色 |

〇方法1 publicにしてUnityで関連付ける

・classの最初に「public Text 名前;」と宣言する

・Start関数の中で「名前.text = “”」とすると最初は表示されない

・変更したいときは適切な位置で「名前.text = “abcde”」とする。「.text」を忘れやすいので注意！

例、スコアの表示

int score = 0;

public Text scoreText;

void Update(){ scoreText.Text = “SCORE：” + score;}

・設定したオブジェクトのInspectorビューの中でスクリプトの中に「名前」が増えていて「None(Text)」になっているのでCanvasに入っているTextの名前にすると反映される

〇方法2 privateにしてスクリプトの中で関連付ける

・書く量は増えるがプログラムの中だけで完結させることができる

・「using UnityEngin.UI」→「private Text 変数名;」→Start関数で関連付ける

・Start関数内では以下のようにする

変数名 = GameObject.Find(“Unityで用いているテキスト名”).GetComponent<Text>();

【Slider】

・スライダー（Slider）は数の大きさを画面上（Canvas）に示すバーである。大きさは基本0から１の間である。

・最初に「using UnityEngine.UI;」と宣言する

例、sliderを半分にする

public Slider slider;

void Start(){

slider.value = 0.5f;

}

・Sliderを作ったときに勝手に生成される「Fill Area」の色を変更するといい感じになる

【Image】

・Imageは画面上（Canvas）に写真を貼り付けるUIである。

例、画面中心に照準を付ける

1. UI – ImageからImageを作る

2. Positionを(0, 0, 0), WidthとHeightを10にする

3. Source ImageをKnobにする

4. Colorを選択する

・スクリプト上で写真をImageに貼り付ける。写真のTexture TypeをSpriteにしておくとスクリプトでSprite型として画像を用いることができる。貼り付けたい写真を「public Sprite 名前;」として宣言して、貼り付けるImageも「public Image 名前;」として宣言しておく。実際に貼り付ける際には「Image型の名前.sprite = Sprite型の名前;」とする

public Image image;

public Sprite sprite;

image.sprite = sprite; //　実際に貼り付ける場所で書く

**〇音**

【音を出す：スピーカー】

・UI – Canvas – Add Componentから「Audio Source」を選択する。一番上のAudioClipに流したい音源を入れる。Play On Awakeにチェックを入れると開始時に音がなる。ループさせたいときはLoopにチェック。

・スクリプトから音を鳴らしたいときは最初に「public AudioSource aus;」と宣言してunityで同期する。音を鳴らすタイミングで「aus.Play();」をすることで設定した音を鳴らせる。

・鳴らす音を変更するには最初に「public AudioCilp 名前;」と宣言してunityで同期する。「aus.clip = 名前;」としてから「aus.Play();」をすると設定した音を鳴らせる。

例、衝突したときのSE

public AudioSource aus; // 自身の名前

public AudioClip coll; // 衝突したときのSE

void OnCollisionEnter(Collision collision){

aus.clip = coll;

aus.Play();

}

【音を拾う：マイク】

・Main Cameraにマイクが付いていて音をひろうことができる。

・Audio ListnerとAudio Sourceを用いる。

[手順]

・Main CameraにAudio Sourceを追加

・Main CameraにSoundManagerという名前でNewScriptを追加

・SoundManagerの中では以下のようにしてマイクを設定数ｒ

**public** AudioClip 音源の変数名;

**public** AudioSource audioSource;

**void** **Start** () {

// AudioSourceを変数に割り当てる

// Unity上でaudioSourceにAudioSourceを手動で関連付けるのと同じ

audioSource = GetComponent<AudioSource>();

}

// ここをダメージ壁から実行して音を鳴らす

**public** **void** **関数名**(){

audioSource.PlayOneShot(音源の変数名);

}

・音を鳴らすObjectのScriptで以下のようにする

public SoundManager soundManager;

void Start(){

// soundManagerという変数に結び付ける。

// ここだけはUnity上で結び付けることはできないのでスクリプトの中で書くこと

soundManager=GameObject.Find(“Main Camera”).GetComponent<”SoundManager”>();

}

void OnCollisionEnter(Collision collision){

soundManager.関数名(); // ここで音を鳴らす

}

・使いたい音源を「wav形式」でAssetsに入れて、Main Camera内のSoundManagerの変数名の中に入れる

**〇考え方**

〇2Dアクション

・当たり判定はMesh Rendererのチェックを外して見えないようにし、isTriggerにチェックを入れて衝突しないようにする。

・異なるオブジェクトの衝突判定には名前で判断しても良いがクローンしたときに面倒なので、タグ名を用いると良い。

・Inspectorビューの最上段でUntagged - Add Tag -「+」からタグ名を入力する。Untaggedを再び選択して入力した名前で設定する。スクリプトの中でタグ名を指定するときには「CompareTag(“タグ名”)」を用いる。if文の中で用いてture or falseとなる

・衝突判定のとき

OnCollisionEnter関数の中で「collision.gameObject.CompareTag」を用いる

OnCollisionEnter(Collision collision){

if(collision.gameObject.CompareTag(“タグ名”)) 処理;

}

・通過判定のとき

OnTriigerEnter関数の中で「other.CompareTag」を用いる

OnTriggerEnter(Collider other){

if(other.CompareTag(“タグ名”)) 処理;

}

・タイマーを作るときはTime.deltaTimeを用いる。これは1フレームごとの経過時間なので変数にUpdate関数で足したり引いたりすればよい。テキストに結び付けるときはtime.ToString(“F2”)とする。F2で小数点以下2桁表示を示す。時間を止めるには「Time.timeScale = 0f;」とする。

・ゲーム終了時に時間を止めたいときは「Time.timeScale = 0f;」とする

〇ブロック崩し

・カメラで撮る映像を奥行きのないものにしたいときはProjectionをPerspectiveからOrthographicに変更する。こうすると平面的な映像になる。

・動く方向が1方向のみのときはFreeze Positionから動きや回転を制限する

・衝突しても動かしたくないときは重さMassを大きくする

・左右のキー入力で動かすときには押した時間で動きが変わるように速度を変化させる。

void Update(){

float speed = 50f;

// キーボードを押した時間によって-1から1までの値をとる

float direction = Input.GetAxisRaw(“Horizontal”);

rb.velocity = new Vector3(direction \* speed, 0f, 0f);

}

・速度の向きと大きさを変えるには「速度 = 単位ベクトル \* 大きさ」の考え方をする。単位ベクトルは「(x, y, z).normalized」で表現する。単位ベクトルにしないと思った通りの大きさにならないことに注意！

・最初は瞬発力を与える必要がある「rb.AddForce(x, y, z, ForceMode.Impulse);」

・衝突した後も速度を変化させないようにするには、常に速度の大きさを監視して変化した瞬間にもとに戻すようにすればよい

ボールのスクリプトの例

public float speed = 20f;

public Rigidbody rb;

void Start(){

rb = GetComponent<Rigidbody>();

Vector3 direction = new Vector3(1f, 1f, 0); // 右ななめ上向き

rb.AddForce(direction.normalized \* speed, ForceMode.Impulse);

}

・ボールが反発するようにボールと壁にPhysic Materialを付け加える。摩擦(Friction)はすべて0, Minimumにして反発(bounciness)を1, Maximumにする。

・増減するObjectの数にアクセスするには同じTagをつければよい。targetというTagの数を扱うには以下のようにする

void Update(){

GameObject[] targets = GameObject.FindGameObjectsWithTag(“target”);

if(targets.Length == 0) Time.timeScale = 0f; // 数が0なら終了

}

○ツムツム

【プレハブ（Prefab）】

・オブジェクトを複数出す、消すということをしたいときはプレハブ（Prefab）にする。プレハブとはオブジェクトの全ての情報をまとめて保存や生成することである。

・最初にHierarchyウィンドウにオブジェクトを作り、できるだけ座標は(0, 0, 0)にする。HierarchyウィンドウのプロジェクトをProjectウィンドウのAssetにドラッグする。このとき、Hierarchyの名前が青色になったら成功となり、Hierarchyウィンドウからは消しても良い。プレハブから生成するときにはAssetからSceneビューやHierarchyウインドウにドラッグアンドドロップする。

・スクリプトではInstantiateを用いて「Instantiate(プレハブ, 位置, 回転);」とする。プレハブはGameObject型である。回転はnew Vector3 (x, y, z)で指定することができないので「Quaternion.identity」か「transform.rotation」とするとよい

・クローン後のRigidbodyを変更するにはInstantiateのGameObjectを用いてRigidbodyを取得する必要がある。

例、原点に角度をつけずオブジェクトを配置して、正面に飛ばす

public GameObject item; // unityで関連付ける

GameObject obj = Instantiate(item, Vector3.zero, Quaternion.identity);

Rigidbody rb = obj.GetComponent<Rigidbody>(); // クローン後のRigidbodyを取得

rb.AddForce(obj.transform.forward \* 1000);

例、元の場所にそのまま配置

Instantiate(item, transform.position, transform.rotation);

・クローンするObject本体に「Instantiate(自分);」としてしまうと無限に生成されることに注意。他のスクリプトかGameManagerから生成するようにするべき

・Instantiateでクローンされたオブジェクトの名前は「名前(Clone)」となってしまうのでStart関数の中で「name = “名前”;」とすると良い

・プレハブで生成するオブジェクトを配下に置きたいときは「Instantiate(プレハブ, 位置, 回転, 親にしたいオブジェクト.transform)」

例、自分の配下に置いて帽子のようにする

Instantiate(item, transform.position + new Vector3(0f, 1f, 0f), Quaternion.identity, gameObject.transform);

・プレハブから作ったクローンは名前が”もとの名前(番号)”となるので衝突や通過判定でそのまま名前を使えないので部分文字列の検索(.Contains(“名前”))を行う。または同じタグをつけてCompareTag(“タグ名”)とする

if(other.gameObject.name.Contains("名前"))

・複雑なゲームを作るときは1つのメインのゲームを動かすプログラムとそのほかの役割を持つスクリプトに分けると管理しやすい。後者のスクリプトは1つにつき1つの仕事をさせるとよい。例えば、ツムの挙動、線を引く，スコアを扱う、ツムの生成、タイマー、マウスのクリックとツムの選択の挙動のような子スクリプトに分けて、メインのスクリプトで呼び出すと良い

**◯いろいろな関数**

【時間を扱う関数】

・Timeから出力される数値はfloatで受け取る

| **呼び出すコード** | **仕様** |
| --- | --- |
| Time.time | ゲーム開始時から経過した時間を取得できる |
| Time.deltaTime | 1フレーム毎の経過時間を取得できる |

例、タイマーを作る

float timer = 0;

void Update(){

timer += Time.deltaTime; // 1フレームの経過時間を1フレーム毎に足す

}

・タイマーの注意点としては経過時間が整数になるとは限らないのでif文の条件を工夫する必要がある。10秒経過をtimer == 10ではなく、10 < timer && 11 < timerとすると良い

【ランダムを扱う関数】

・「Random.Range(min, max)」とするとminからmaxまでのfloatかintの値を返す。intのときはPythonと同じでmax-1が最大値となることに注意！

例、サイコロ

int sai = Random.Range(1, 7); // 1から6

【GetComponent】

・スクリプトでComponentを取得したいときはunityで同期させてもいいが、GetComponentを使うこともできる。

・「操作したいオブジェクト.GetCoponent<Componentの名前>()」で取得できる

例、SphereのRigidbodyを取得する

public GameObject sphere;

Rigidbody rbSphere;

void Start(){

rbSphere = sphere.GetComponent<Rigidbody>();

}

・自身のComponentを取得するときは「GetComponent<Componentの名前>()」として、先頭を省略することができる

例、Rigidbodyを同期させる

Rigidbody rb; // publicはいらない

void Start(){

rb = GetComponent<Rigidbody>();

}

**〇その他**

・異なるScript同士で値を共有したいときは、Scriptの中で「public Script名 変数名;」で宣言することでプログラムの中からUnity上で紐づけられるようになる。Unity上で直接手動で結び付けるのと変わらないはず。Start関数の中で以下のようにする

変数 = GameObject.Find(“オブジェクト名”).GetComponent<Script名>();

ここでオブジェクト名の中にScriptが入っているようにすること。

これで「Scriptの変数名.使う変数名」とすることでpublicにしてある変数が使えるようになる。

例、タイムを共有する

**public** TimerScript timer;

**void** **Start**(){

// TimeというUIの中にあるTimerScriptにアクセス

// Unity上でtimerの中にTimeScriptを入れるのと変わらない

timer = GameObject.Find("Time").GetComponent<TimerScript>();

}

**void** **OnCollisionEnter**(Collision collision){

// TimerScriptの中にあるtimeという変数にアクセス

timer.time = timer.time - 10f;

}

・スクリプトでMaterialを変更したいときは先頭でMaterial型をpublicで宣言してから、GetComponent<MeshRenderer>().materialにMaterialをぶちこむ

public Material[] 変数名 = new Material[用いる数]; // Material格納

GetComponent<MeshRenderer>().material = ColorSet[0]; // 貼り付けたいMaterial

・スクリプト上の値をunity上で変更したいときは変数宣言時に前に[SerializeField]を付与

[SerializeField] float jumpPower;

unity上での変更の意味ではpublicとほとんど同じ挙動をするが、他のスクリプトから参照が不可である点が異なる。publicなら他のスクリプトから変更可能。

・重力の値を変更したいときは「public Rigidbody rb;」をしてunityでRigidbodyと

rbを結びつける。「[SerializeField] Vector3 localGravity;」と宣言してunityから値を変更できるようにする。Update関数の中で「rb.AddForce(localGravity, ForceMode.Acceleration);」として加速度を変更させる。

・1つのスクリプトを複数のオブジェクトにつけてある一つだけを変更したい際には、変数宣言の際にpublicを付けて、unityで値をそれぞれ決めると良い

例、シューティングゲームの自分と敵の弾の向き

「public float speed;」と宣言して、unity上で自分は上向きに射出したいので正の値を、敵は下向きに射出したいので負の値を設定すれば良い

**〇注意**

・変数宣言の前に着ける「private」は値をScriptの中でのみ使うという意味で、「public」は他のScriptやUnityでも値を変更することを指す。publicにしたときはUnityのタブにある値が最も優先されるのでScriptで値を変更しても意味がない。タブの値を変更するかprivateにする。privateにしたらそもそもUnityのタブには出てこなくなる

・UIテキストを使うときに「名前.text」の.textを忘れやすいので注意！

・Rididbodyをコードで変更したときにオブジェクトに反映させるのを忘れやすい！

・OnTriggerEnterの引数の型はCollisionではなくColliderになることに注意！

**○2Dゲーム**

・2Dゲームだとカメラがデフォルトで遠近感がないモードになっている。z軸はObjectを手前に配置するか奥には位置するかに使う。基本Main Cameraはz = -10から大きくなる方向へ撮っているので-10に近いと手前に配置される。

・オブジェクト設置には2D Object → Spriteを用いる。Spriteとは2Dの画像型オブジェクトのことである。

・3Dオブジェクトのときは必要なかったが、2Dオブジェクトであたり判定をつけるにはAdd Component → PolygonColider2Dをつける必要がある

・2D Object → SpriteShade → Closed Shadeを用いると点を動かして自由な形のスプライトを作ることができる。InspectorウィンドウからEdit Splineをクリックすると点の追加や移動ができるようになる。

・SpriteShadeに画像を貼り付けるには、ProjectウィンドウでCreate → 2D → Sprite Shade Profileを選択して、InspectorウィンドウのTextureに画像をドラッグする。このProfileをSpriteShadeにつけることで画像を貼り付けることができる。画像を繰り返しにしたいときは、写真を選択してInspectorウィンドウのWrap ModeをRepeatにする

・マウスカーソルと2Dオブジェクト衝突判定をしたいときはレイ(Ray)というレーザービームのようなものを用いて以下のようにする。

**string** **getHitName**(){

//マウスが押されているところをワールド座標系に変換

Vector3 mousePosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);

//マウスのワールド位置からRayを飛ばす

Ray ray = Camera.main.ScreenPointToRay(mousePosition);

//マウスのワールド位置から、カメラの撮影方向（xyは0で、zが1）で、長さは

未指定(永遠)で当たる物ないか検索

RaycastHit2D hit = Physics2D.Raycast(mousePosition, **new** Vector3(0, 0, 1));

//何にもあたってない時はnull。nullを返す

**if** (hit.collider == null) **return** null;

//何かにあたっているなら、あたっているオブジェクトの名前を返す

**return** hit.collider.gameObject.name;

}

・マウスカーソルを線で引くにはLineRendererを用いる。これは点と点の間に線を引く機能である。点を複数個追加することによって線に見えるようになる。Create EmptyしてAdd Component → LineRendererとする。同じくNew Scriptを追加してLineRendererに点を加えるプログラムを書く

**void** **Start**(){

line.material = **new** Material(Shader.Find("Sprites/Default"));

}

void Update(){

//LINEを描いていなければ何もしない

**if**(isDrawing == false) **return**;

//マウスの位置を取得して、ポイントの位置にする

Vector3 newPointPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);

//x,yはマウスの位置で、zはLineのオブジェクトの位置にする

newPointPos.z = transform.position.z;

//直前のマウスの位置とめちゃくちゃ近かったら追加しない

**if** (Vector3.Distance(lastPos, newPointPos) > 0.1){

//直前のマウスの位置に今のマウスの位置をいれる

lastPos = newPointPos;

//ポイント数を1つとしてLineRenderに設定する

addCount++;

line.positionCount = addCount;

line.SetPosition(addCount - 1, newPointPos);

}

}

*//ライン描画開始*

**public** **void** **StartDraw**(){

*//全てクリアしてLineを描いているフラグをtrueにする*

line.positionCount = 0;

lastPos = Vector3.zero;

addCount = 0;

isDrawing = true;

}

*//ライン描画終了*

**public** **void** **StopDraw**(){ isDrawing = false;}

衝突する方のテキストでは以下のようにする。

**void** **Update**(){

*//マウスをクリックしはじめたらにtrueになる*

**if** (Input.GetMouseButtonDown(0)){

onDownBegin();

}

*//マウスが押されている間はtrueになる*

**else** **if** (Input.GetMouseButton(0)){

onDown();

}

**else** **if** (Input.GetMouseButtonUp(0)){

onUp();

}

}

*//押され始めた時の処理*

**void** **onDownBegin**(){

**string** objName = getHitName();

**if** (objName == "Start"){

isDoing = true;

messageText.text = "スタート";

line.StartDraw();

}

}

*//押されている間の処理*

**void** **onDown**(){

**if**(isDoing == false) return;

**string** objName = getHitName();

*//なにかと衝突した時だけそのオブジェクトの名前をログに出す*

**if** (objName == null){

messageText.text = "アウト";

isDoing = false;

line.StopDraw();

}

**else**{

**if** (objName == "Goal"){

messageText.text = "ゴール";

isDoing = false;

line.StopDraw();

}

}

}

// クリックをやめたときの処理

**void** **onUp**(){

**if** (isDoing == false) **return**;

messageText.text = "アウト";

isDoing = false;

line.StopDraw();

}

・一定間隔で動くオブジェクトを配置したいときはVector2.Lerp(始点, 終点, 0～1の比率)を用いる。始点と終点はどちらもVector 2Dであり、座標を示す。比率は0のときに始点にあり、1のときに終点にある。間にあるときは良い感じの位置にいる。全てpublicにしてclassの前に[ExcecuteInEditMode]にすることでUpdate関数の中身を編集中も実行することができる。unity上で比率をいじると編集中でも移動しているのが確認できる。

[**ExecuteInEditMode**]

**public** **class** **MoveWall** : **MonoBehaviour**{

**public** Vector2 startpos;

**public** Vector2 endpos;

**public** **float** startEndRate = 0;

**void** **FixedUpdate**(){

*//startEndRateが0以下ならstartposに、1以上ならendposに、その間なら比率*

*に応じた位置を自分の位置にする*

transform.position = Vector2.Lerp(startpos, endpos, startEndRate);

}

}

・一定間隔で行ったり来たりを繰り返すにはTime.time関数を1で割ったあまりを用いて小数点を取り出して、0.5以下のときは行き、0.5以上のときは帰りにすれば良い

float t = Time.time % 1;

if(t < 0.5) startEndRate = t \* 2; // 行き

else startEndRate = (1f - t) \* 2; // 帰り