GameProgramming

09 Animation

CHARACTER ANIMATION

- Unityでキャラクターデータを扱う
  - キャラクターのモデルは外部ソフトウェアで作成
    （Mayaで作る場合はHumanIKの使用を推奨，**FBX**形式推奨）

  - キャラクターモデルは
    「**形状データ**」と「**骨格データ**」を持つ
    「**アニメーションデータ**」を持つ場合もあり.

  - インポート後はRigを「**Humanoid**（人型）」に変更
    （Projectウィンドウ内でキャラクターファイルを選択）
    人型の汎用モーション（他者作成など）の利用が可能になる.

UNITY-CHAN UCL2.0

- 本日使用するファイルにはUnity-Chanが含まれます.
  「GP08_01.unitypackage」「GP08_02.unitypackage」

弊社キャラクターのデジタルアセットデータを再配布する場合

弊社キャラクターのデジタルアセットデータを、git等の共有WEBサービスを使って再配布する場合、ライセンスロゴもしくはライセンス表記の掲示に加えて、ライセンス関連ファイル一式を同梱して配布するようにしてください。ライセンス関連ファイル一式は、こちらよりダウンロードできます。

- 配布パッケージにライセンスファイルが
  含まれています.ご一読ください.

  Asset > License

キャラクターの準備　1

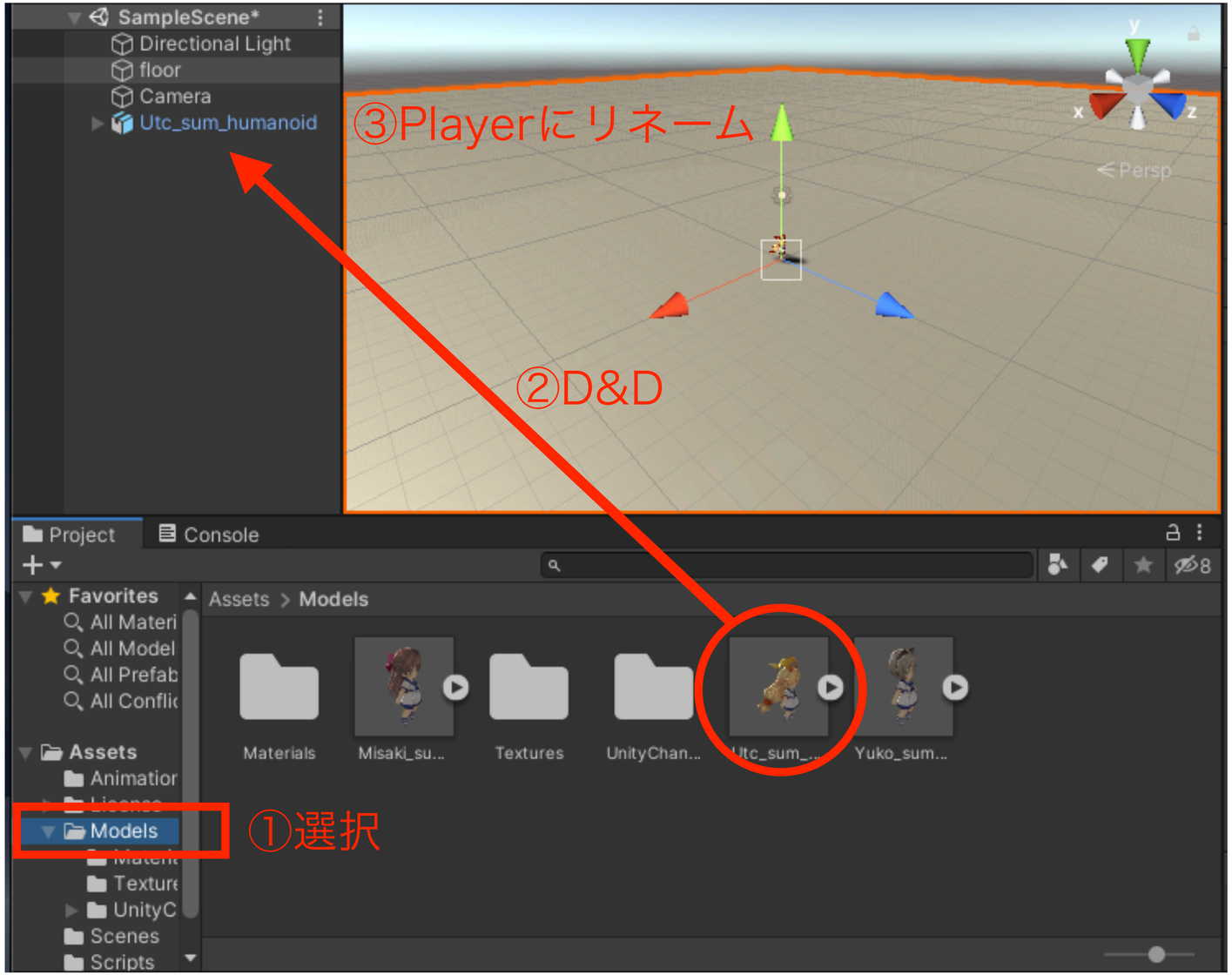
- プロジェクトを新規作成

- 配布データ（GP08_01.unitypackage, GP08_02.unitypackage）をFinder上で
  ダブルクリックしてimport

- 床を作る **3D Object > Plane**
  インスペクター設定
  Floorにリネーム
  scale x:20，y:1，z:20
  Rigidbodyを追加　Use Gravity:off，Is Kinematic:on

- ProjectウィンドウのModelsフォルダ内の
  「Utc_sum_humanoid」をHierarchyウィンドウにD&D
  「**Player**」にリネーム（後のスクリプトで使用するので大文字小文字の区別も大事）
  TagをPlayerに変更

キャラクターの準備　1


SampleScene*
Directional Light
floor
Camera
Utc_sum_humanoid
③Playerにリネーム
②D&D
Persp
Project
Console
Favorites
All Materi
All Model
All Prefab
All Conflic
Assets
Animatior
Models
Texture
UnityC
Scenes
Scripts
Assets > Models
Materials
Misaki_su...
Textures
UnityChan...
Utc_sum_...
Yuko_sum...
①選択

キャラクターの準備　2

- PlayerにRigidbodyを設定
  Constraints内の**Freeze Rotation XYZ** 全てにチェック
  （チェックを入れておかないと力が加わった際に転ぶ）

- PlayerにPhisics>**CapsuleCollider**を設定
  sceneビューの緑色のコライダーを確認しながら
  center Y, Radius, Heightを設定（centerYはHeightの半分の値）
  例）**Center Y:0.5**
  Radius:0.3, Height:1.0

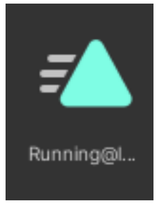
- Main CameraをPlayerの子の階層に（PlayerにD&D）
  MainCameraの位置を調整してPlayerをすぐ後ろから見るように

アニメーションの設定

- Animator Controllerを作成（Projectウィンドウの＋）
  リネーム「PlayerController」
- 作成した「PlayerController」をPlayerにD＆D

- PlayerControllerをダブルクリック(Animatorウィンドウが開く)

- Project ウィンドウ内の検索窓でStandingと入力
  「Standing@loop」をAnimatorウィンドウにD&D
- Project ウィンドウ内の検索窓でrunと入力
  「Running@loop」をAnimatorウィンドウにD&D

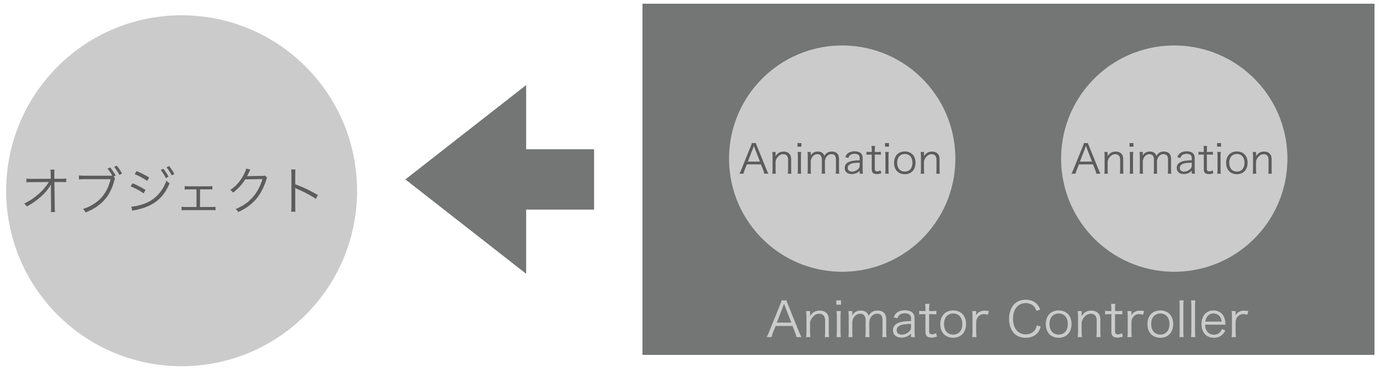


以下@loopを省略して記載します

TIPS

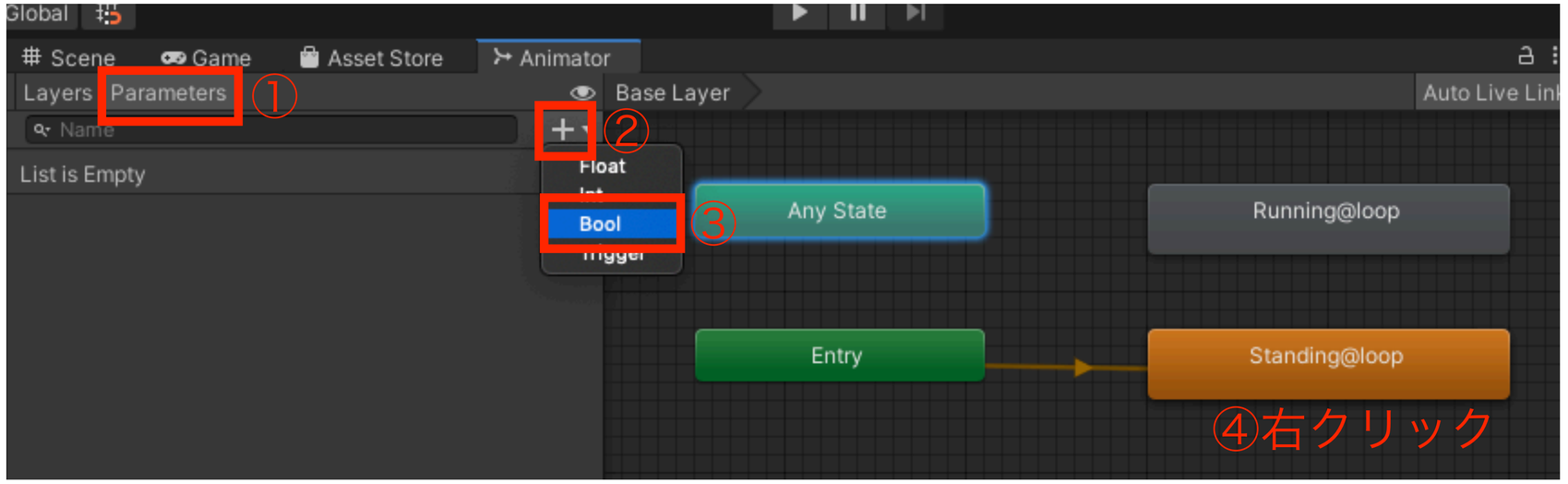
- Animationはオブジェクトの個々のアニメーションデータ
  （移動，回転，拡大縮小，歩く，走るなど）
- Animator Controllerは
  上記Animationを組み合わせて遷移を設定する場所

  オブジェクトにAnimationを割り当てたい場合，
  AnimatorControllerを経由する



アニメーションの設定

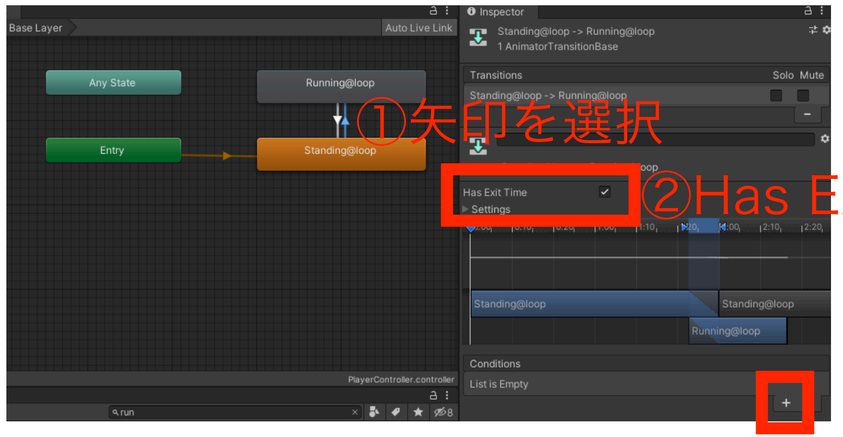
- Animatorウィンドウ内のタブ「Parameters」を選択
  検索窓右の＋ボタンをクリック，Boolを選択，変数名をRunにする
- オレンジ色の「Standing」 を右クリック
  Make Transtionを選択　「Running」まで矢印を伸ばして選択
  「Running」から「Standing」にも同様に矢印を作る



アニメーションの設定

- 「Standing」から「Running」に伸びる矢印を選択,
  inspector 内のHas Exit Timeのチェックを外す
  conditionsの＋ボタンをクリック　Run，trueに設定

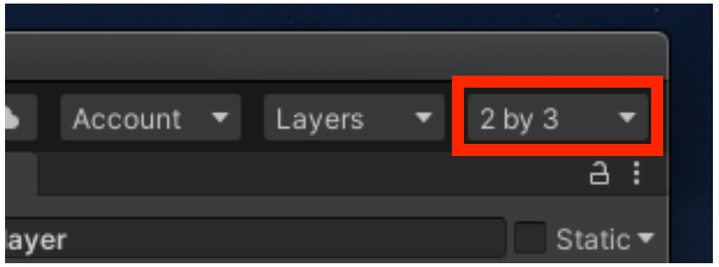
- RunningからStandingに伸びる矢印を選択,
  inspector 内のHas Exit Timeのチェックを外す
  conditionsの＋ボタンをクリック　Run，falseに設定



Has Exit Time がONの場合
アニメーションファイルの終了まで待った後，遷移

アニメーションの設定

- 動作の確認
  画面レイアウトを2by3にする.
  プレイボタンを押して,



Unityのウィンドウ右上

①スタンディングポーズを取るか？
（初期のAスタンスポーズから変化すればOK）

②Animatorウィンドウの
（表示されていない場合はPlayerContorollerをダブルクリック）
Runにチェックを入れて走り出すか？

③Animatorウィンドウの
Runのチェックを外し止まるか？

playerControl.cs (プレイヤーキャラクターに適用)

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

public class PlayerControl : MonoBehaviour
{
    public float forwardSpeed = 2.0f;   // 前進速度
    public float rotationSpeed = 100.0f;  // 回転速度
    private Animator anim; //Animator型変数宣言
    private bool runFlg;

    void Start()
    {
        anim = GetComponent<Animator>(); //このスクリプトがアサインされたキャラクターのアニメーターコントローラーを取得

    }

    void Update()
    {
        float v = Input.GetAxis("Vertical"); //上下キーの取得  Up:1, Down:-1  に段階的に変化
        float h = Input.GetAxis("Horizontal"); //左右キーの取得 Right:-1 Left:1  に段階的に変化

        if (v > 0.1 || v < -0.1 || h > 0.1 || h < -0.1)
        {
            runFlg = true;
        }else
        {
            runFlg = false;
        }
        anim.SetBool("Run", runFlg); //アニメーターコントローラーのRunに値(runFlg)を代入
        transform.position += transform.forward * forwardSpeed * v  * Time.deltaTime; //プレイヤーを移動
        transform.Rotate(0, rotationSpeed * h * Time.deltaTime, 0);  //プレイヤーを回転
    }
}
```

playerControl.cs (プレイヤーキャラクターに適用)

```
transform.position += transform.forward * forwardSpeed * v  * Time.deltaTime;
```

transform.position +=：現在の位置に右辺の値を加える（移動量：速度×時間）

Transform.forward：オブジェクト正面方向の長さ１のベクトル（向き）
forwardSpeed：変更可能なスピード（float）
v：上下キーの入力（下：-1, 上：1, 未入力:0）
Time.deltaTime：1フレームあたりの時間（直前のフレームと今のフレーム間で経過した時間[秒]）

```
transform.Rotate(0, rotationSpeed * h * Time.deltaTime, 0);  //プレイヤーを回転
```

敵キャラクター

- ProjectウィンドウのModelsフォルダ内の
  「Misaki_sum_humanoid…」をHierarchyウィンドウにD&D
  Enemyにリネーム

- RigidbodyとCapsuleColliderをプレイヤーキャラと同様に設定

- Animator Controllerを作成（名前はRun）し，EnemyにD＆D

- Project ウィンドウ内の検索窓でrunと入力
  「Running@loop」をAnimatorウィンドウにD&D

enemyControl.cs （enemyに適用)

```
public class enemyControl : MonoBehaviour
{
    private Vector3 velocity;// キャラクターの移動量
    private Transform target; //Transform情報を入れておく変数
    public bool escape = false; //モードの切り替え用ブール変数（false:追ってくる，true:離れる）
    public float speed = 1.0f;

    void Start()
    {
        //指定したゲームオブジェクトのTransform情報取得
        target = GameObject.Find("Player").transform;
    }

    void Update()
    {
        if (escape == true)  //プレイヤーから離れる設定の場合
        {
            //プレイヤー（target）の位置から自身の位置に向かうベクトル
            Vector3 vec = transform.position - target.position;
            //自身の向きを，現在の向きからvecの向きに緩やかに変更
            transform.rotation = Quaternion.Slerp(transform.rotation,
                Quaternion.LookRotation(new Vector3(vec.x, 0, vec.z)), 1.0f);
        }
        else    //プレイヤーを追ってくる設定の場合
        {
            //プレイヤー（target）の方を向く
            transform.LookAt(target);
        }
        // 以下、キャラクターの移動処理(playerControlと同様)
        velocity = new Vector3(0, 0, speed);
        velocity = transform.TransformDirection(velocity);
        transform.localPosition += velocity * Time.fixedDeltaTime;
    }

}
```

課題

- 鬼ごっこをベースにしたキャラクターアクションゲームを制作せよ

- ステージの制作　少なくとも床と壁を制作
  （自作・フリーアセットの使用可）

- プレイ動画を撮影し,
  ファイル名を「学籍番号_GP08」として提出せよ.

- 余裕がある人は次ページ以降の応用課題に挑戦してください

- 来週もこの続きを行うのでプロジェクトフォルダを消さないこと

応用課題

- ゲームクリア，ゲームオーバーの実装
  enemyControl内に，Playerタグを持つオブジェクトに
  ぶつかった時の処理を記述
  北坂先生担当回を参考にすること

- アニメーションを追加
  ジャンプアニメーションを追加し，
  スペースキーを押すとジャンプモーションを行う

  様々な手法があるので，紹介する方法以外でもOK
  以降で紹介する方法は
  ・ジャンプモーション中に重力を切る方法
  　ジャンプ中に下には落ちなくなるが，障害物乗り越えは×
  　スムーズなジャンプアニメーションを再生できるメリット

応用編：ゲームクリアとゲームオーバー

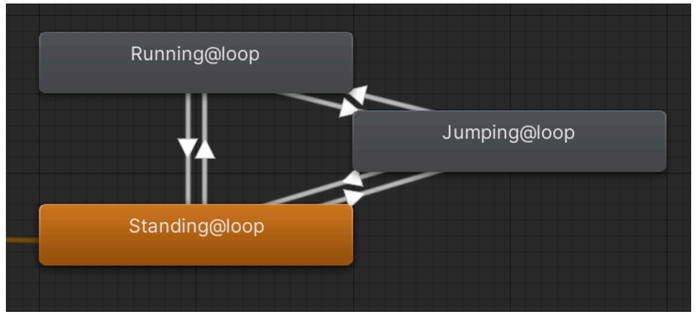
- ゲームクリア，ゲームオーバーの実装　以下のどちらかを実装
  - 逃げる敵を捕まえるとゲームクリア
  - 追ってくる敵に捕まるとゲームオーバー

- enemyControl.cs に以下を追加する.

```
void OnCollisionEnter(Collision obj){
    if (obj.gameObject.tag == "Player")  {
        Debug.Log("  GameClear Or GameOver ");
    }
}
```

応用編：キャラクターをジャンプさせる

- PlayerControllerをダブルクリック
- Project ウィンドウ内の検索窓でjumpと入力
  再生マークのアイコンの「Jumping@loop」をAnimatorウィンドウにD&D（runningの右あたり）
- Animatorウィンドウ内のタブ「Parameters」を選択
  検索窓右の＋ボタンをクリック，Boolを選択，変数名をJumpにする
- Runnningを右クリック　Make Transtionを選択　Jumpingまで矢印を伸ばして選択
  Standingを右クリック　Make Transtionを選択　Jumpingまで矢印を伸ばして選択
  JumpingからRunnning，JumpingからStandingにも同様に矢印を作る
- RunnningからJumpingに伸びる矢印を選択，
  inspector 内のHas Exit Timeのチェックを外す
  conditionsの＋ボタンをクリック
  Jump，trueに設定



- **Jumpingから伸びる矢印はHas Exit Timeのチェックを外さない**
  JumpingからRunnningに伸びる矢印を選択，conditionsの＋ボタンをクリック Run，trueに設定
  JumpingからStandingに伸びる矢印を選択，conditionsの＋ボタンをクリック Run，falseに設定
  （ジャンプのアニメーションを最後まで再生し，その後条件に従い遷移する）

応用編：キャラクターをジャンプさせる

PlayerControl.cs スクリプトの修正

void start(){}の上部　変数宣言に　private Rigidbody rigb;　を追加
void Start(){　}の中に　rigb = GetComponent<Rigidbody>(); を追加

void Update(){　}の中に以下を追加

```
//スペースを押した時　アニメーター内のブール変数Jumpをtrueに
if (Input.GetKey(KeyCode.Space)) {  anim.SetBool("Jump", true); }

//現在キャラクターが行なっているアニメーション名を取得
AnimatorStateInfo state = anim.GetCurrentAnimatorStateInfo(0);
if (state.IsName(“Jumping@loop”))  //Jumpingアニメーション中のとき
 {
     rigb.useGravity = false;                  //重力をoffに
     anim.SetBool("Jump", false);    //Jumpをfalseに
}else                                          //Jumpingアニメーションでないとき
{
   rigb.useGravity = true;                   //重力をonに
}
```