Unity 2 D 講習

0．Unity 2 D

今回からは Unity で 2 D のゲームを作ってもらいます。とはいえ基本は 3 D のゲームと変わりません。シーンやオブジェクトの考え方は変わりませんし、プレハブ化も Instantiate もできますし、AddForce すれば動きます。しかし、一部のコンポーネントや命令が変わるほか、描画の方法が「スプライト」というものに変わります。これからの講習は、そういったことについて教えていきます。

1．プロジェクトの作成

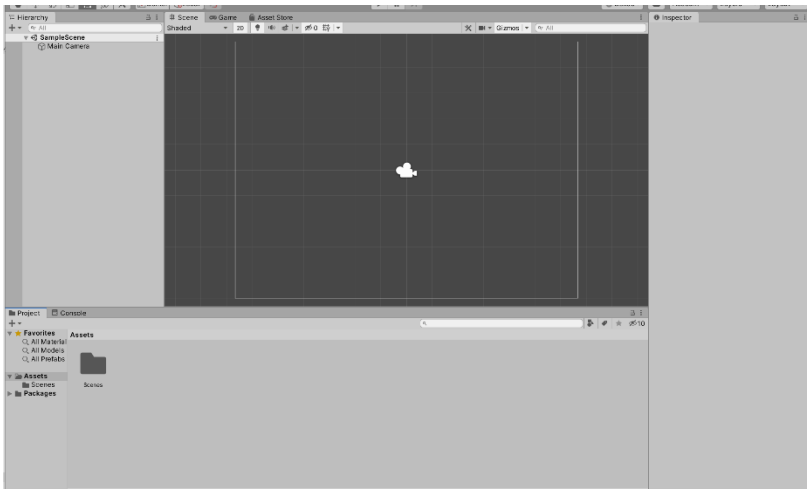
2 D のプロジェクトを作成します。

① Unity Hub で新規作成ボタンを押します。

② テンプレートを「2 D」にして、適当な名前をつけて、プロジェクトを作成します。

③ しばらく待ちます。

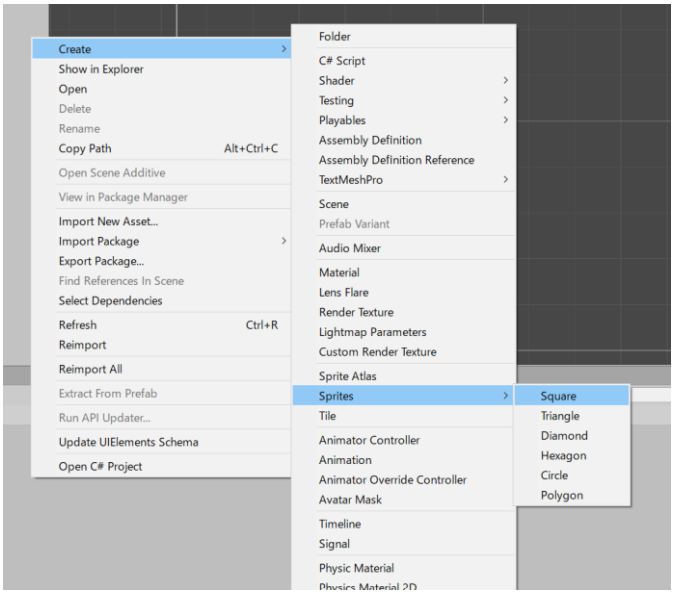
④ 2 D のプロジェクトができます。3 D とはすこし違うのが分かります。



2．スプライト

スプライトは、画像を張り付けるための２D のオブジェクトです。まずは作ってみましょう。

① プロジェクトビュー上で右クリックし、Create>Spreites>Square を選択します。名前の入力を求められますが、そのままにしてください。



すると白いものができます。これがスプライトの元となる画像です。

② 今の画像をシーンにドラッグアンドドロップします。すると白い四角がシーンにあらわれます。これで、シーンに白い四角の画像が張り付けられたオブジェクト＝スプライトができました。

3．外部ファイルの取り込み

今まで、Unity 内部にデフォルトで存在するデータを使ってゲームを作ってきました。これも立派なゲームですが、プレイヤーがただの球体とか四角だと味気ないです。そこで、外部からキャラの絵のデータを取り込み、かわいいキャラを動かせるようにしましょう。

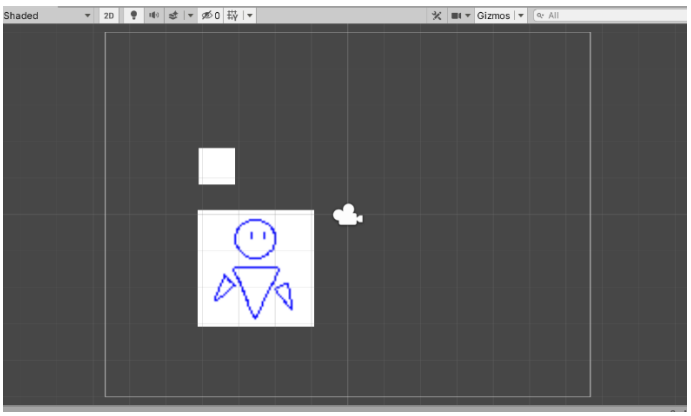
① 画像を用意します。Windows 標準のペイントっていうソフトで書けるので適当に描いてください（タスクバーの検索窓にペイントって打ったらでてきます。使い方はシンプルなのでここでは書きません。わかんなかったら聞いて）。描くのがだるいって人は僕に言ってくれたら一応素材あげます。また、自分の PC でいいソフト持ってるぜって人はぜひそっちを使ってください。

② 自分で描いた人はわかりやすい場所にその絵を保存してください。このとき、ファイル名に日本語を使うとバグる可能性があるのでアルファベットでにしてください。

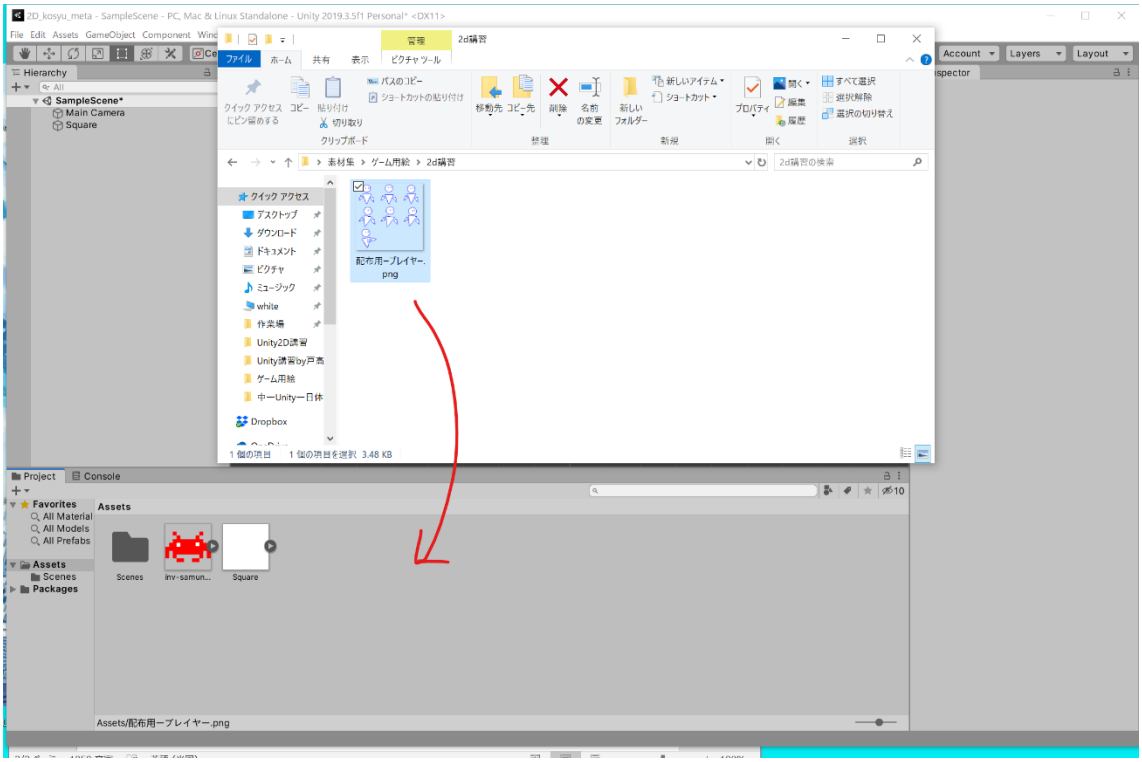
③ 画像を調達する方法を書いておきます。飛ばしてください。

方法は主に４つあります。
１：ネットでフリー素材を検索する（著作権には注意）。
２：自分で描く。
３：人に頼む。
４：Unity の AssetStore を使う。（素材を販売している場所。有料のものが多い）。
飛ばしてください終わりです。

画像を描いたら、背景を透過させます。「手軽に透明 png」なるソフトをあげるので、それを使ってください（ここまで来たら僕を呼んでください）。なぜ透過させるかというと、そうしないと以下のように白い四角ができてしまうからです。ペイント以外のツールで自分で描いた人はそういう設定をして保存しましょう。



④ 画像を保存したフォルダを PC から開き、画像をプロジェクトビューにドラッグアンドドロップします。

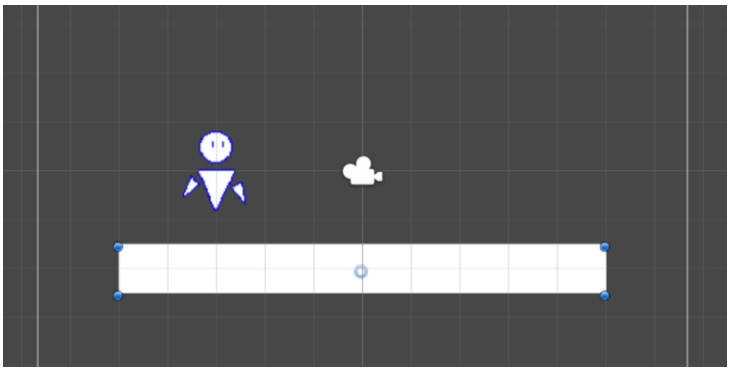


⑤　その画像をプロジェクトビューからシーンにドラッグアンドドロップします。

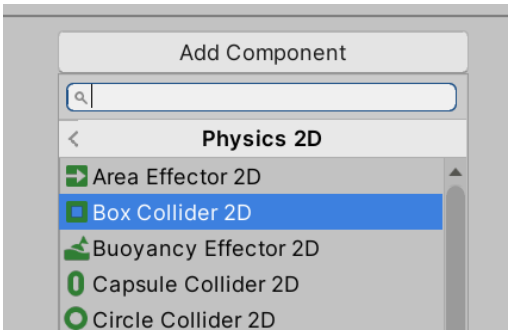
4．準備をする

今回作るゲームは、初回の一日体験で作ったような横２Ｄアクションにします。まずはフィールドを作りましょう。

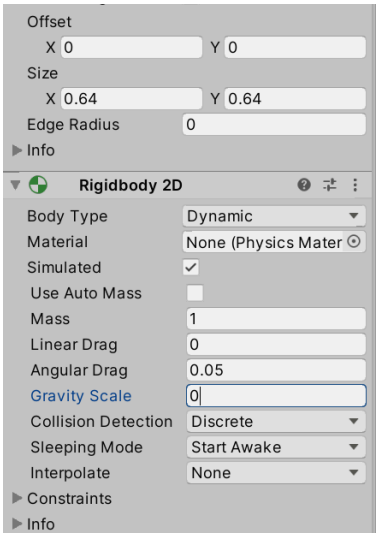
①　スクリーンショットのように白い四角とプレイヤーを配置します。プレイヤーの座標は（-3,0)、大きさは（3,3)、四角は座標（0,-2)、大きさ（10,1）です。



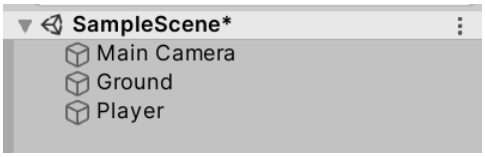
②　当たり判定をつけます。四角とプレイヤーの両方に、Add Component から Physics2D→BoxCollider2D を選択して当たり判定をつけてください。２Ｄがついていることに注意してください。



③ プレイヤーに Add Component→Physics2D→Rigidbody2D を選択して Rigidbody2D コンポーネントを追加してください。こいつは Rigidbody の 2 D 版です。そのうえで Gravity Scale を 0 にしてください。これは重力を働かせないということです。重力は後で自前で実装します。



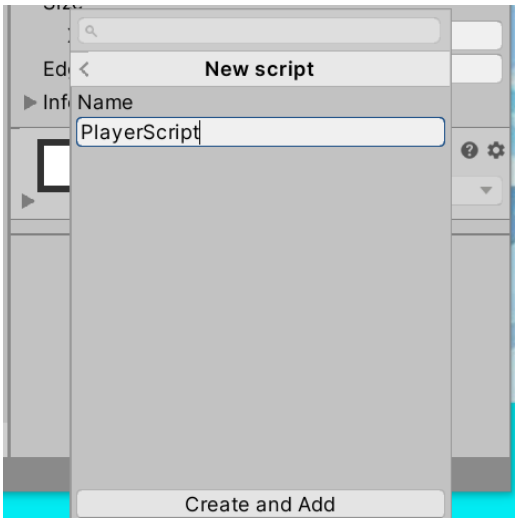
④ オブジェクトの名前を変更しましょう。Square は Ground に、取り込んだ画像は Player に名前を変えてください。



5．プレイヤーを動かす

プレイヤーを動かしましょう。

① Player に Add Component→New Script で、PlayerScript という名前のスクリプトを追加してください。



② 以下のように打ち込んでください。

```
Assets > PlayerScript2.cs > ...
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    0 references
5   public class PlayerScript2 : MonoBehaviour
6   {
        2 references
7       private Rigidbody2D rigid;
        2 references
8       private float XForce;
        1 reference
9       public float Speed;
10      // Start is called before the first frame update
        0 references
11      void Start()
12      {
13          rigid = GetComponent<Rigidbody2D>();
14      }
15
16      // Update is called once per frame
        0 references
17      void Update()
18      {
19          XForce = Speed * Input.GetAxis("Horizontal");
20          rigid.velocity = new Vector2(XForce,0);
21      }
22  }
23
```

GetComponent<>();：移動の際に使っていたプログラムの一部ですが、これの本当の意味は「コンポーネントを取得する」です。<>の中に取得したいコンポーネントの名前を入れると、そのコンポーネントの情報を取得できます。() を書き忘れないようにしましょう。

Input.GetAxis(“Horizontal”)：右矢印キーが押されていたら１、左矢印キーが押されていたらー１、どちらも押されていなければ０を返す命令。滑らかに数値が変化するので、移動の「だんだん速くなる」感が出ます。

rigid.velocity = new Vector2(XForce,0)：rigidbody2D コンポーネントが持つ情報の中の、速度（velocity）を(XForce,0)にする、という意味。Vector2 は Vector3 の仲間で、２つの数値情報を持ちます。

③ インスペクターから Speed を適当に（5 くらい）編集して実行すると、横移動ができるようになります。

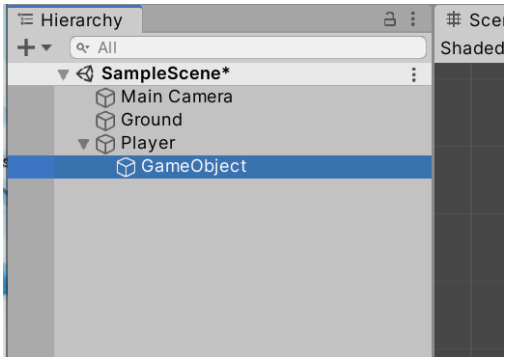
6．着地判定

次は着地ができるようにします。このためには地面にあたっているかを判断することが必要です。いくつか方法がありますが、今回は接地判断用のオブジェクトを作ってそれに判断をさせます。

① ヒエラルキーで右クリック→Create Empty をクリックします。すると GameObject という名前のオブジェクトができます。こいつには Transform 以外のコンポーネントがついていません（ので形もありませんし見えません）。このオブジェクトはゲームの管理によく使います。

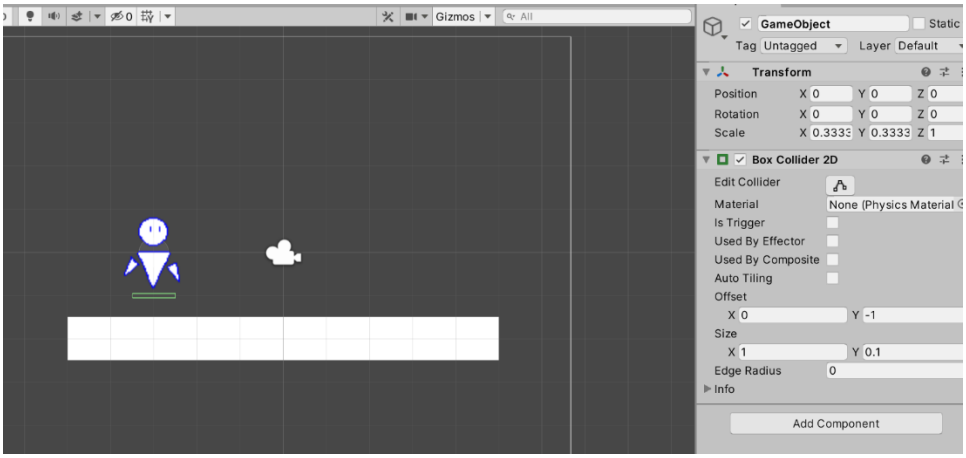
② 子オブジェクトというものを教えます。子オブジェクトは親となるオブジェクトとの位置関係が固定されます。つまり、親が動くと子もついてくるということです。ほかにも関連づいたオブジェクトを親子にすることで、見やすく管理しやすくなるというメリットがあります。

③ ヒエラルキー上で、GameObject を Player にドラッグアンドドロップしてください。すると以下のようになると思います。

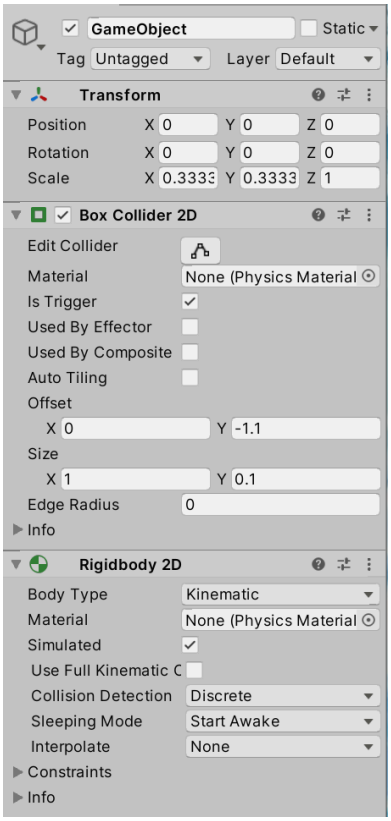


これは GameObject が Player の子オブジェクトとなったことを表します。

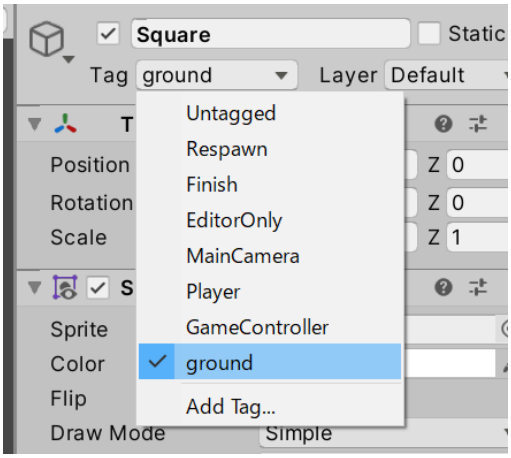
④ Add Component から Box Collider 2D を追加してください。そして、Player の下の適当な位置と大きさに Collider を移動変形してください。Offset と Size をいじればできます。このとき・Player の当たり判定と当たらないような位置・Player の当たり判定の横の長さと同じ横の長さにしてください。また、Is Trigger にチェックを入れてください。



ついでに Add Component→Physics2D→Rigidbody2D で Rigidbody 2 D コンポーネントを追加してください。追加したら BodyType を Kinematic にしてください。これを付けると重力などの影響を受けないようになります。



⑤ Ground の Tag を変更して ground にします。Add Tag から Tag を追加する方法は 3 D 講習資料を見てください。



⑥ Add Component→New Script から LandScript という名前でスクリプトを付けてください。中身は以下のように。

```
public class LandScript : MonoBehaviour
{
    // Start is called before the first frame update
    0 references
    void Start()
    {

    }

    // Update is called once per frame
    0 references
    void Update()
    {

    }

    3 references
    public bool IsLand;
    0 references
    void OnTriggerEnter2D(Collider2D Hit)
    {
        if(Hit.CompareTag("ground")){
            IsLand = true;
        }
    }
    0 references
    void OnTriggerExit2D(Collider2D Hit)
    {
        if(Hit.CompareTag("ground")){
            IsLand = false;
        }
    }
}
```

Bool 型の変数 IsLand を定義し、足場にタグが ground の当たり判定が入ったらそれを true に、出たらそれを false にするという意味になります。
OnTriggerEnter2D(Collider2D Hit)：OnTriggerEnter の 2 D 版。
Hit.CompareTag(“ground”)：当たったもののタグが ground なら、の意味。

OnTriggerExit2D(Collider2D Hit)：OnTriggerEnter2D の仲間。Enter の方は当たり判定に何かが入った時に処理が行われますが、こっちは当たり判定から何かが出た時に処理されます。ほかに Stay っていうのもあるけど今は無視。

7．着地とジャンプ

着地しているかの判断ができるようになったので、着地していなければ重力を働かせ、していれば下に動かさないというプログラムを書いていきます。この機能は Rigidbody にありますが、物理演算は予期せぬ動作の温床となる他、自由度が低いので、演算機能を切って手動で重力を実装します。

① PlayerScript を書きかえます。

```
5   public class PlayerScript2 : MonoBehaviour
6   {
        2 references
7       private Rigidbody2D rigid;
        2 references
8       private LandScript landSc;
        2 references
9       private float XForce;
        3 references
10      private float YForce;
        1 reference
11      public float gravity;
        1 reference
12      public float Speed;
13      // Start is called before the first frame update
        0 references
14      void Start()
15      {
16          rigid = GetComponent<Rigidbody2D>();
17          landSc = GetComponentInChildren<LandScript>();
18      }
19
20      // Update is called once per frame
        0 references
21      void Update()
22      {
23          XForce = Speed * Input.GetAxis("Horizontal");
24          if(landSc.IsLand){
25              YForce = 0;
26          }else{
27              YForce -= gravity;
28          }
29          rigid.velocity = new Vector2(XForce,YForce);
30      }
31  }
32
```

private LandScript：LandScript を変数の型として扱っています。Unity ではすべてのコンポーネント（の名前）を型にできるので、自作スクリプトの名前も型にできます。

GetComponentInChildren<>()：子オブジェクトについているコンポーネントを

取得する命令です。

LdScript.IsLand：LdScript の中には LandScript の情報が入っています。そこにはさらに IsLand という bool 型の変数の情報も入っています。ここではそれを取り出しています。

if(～){A}else{B}：～だった時は A、～ではなかった時は B の処理を実行します。

② ジャンプできるようにします。さらに PlayerScript を編集します。

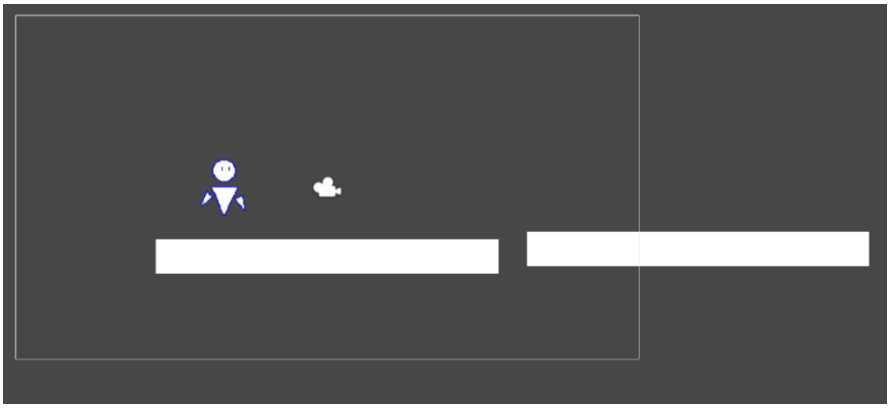
```
    private float XForce;
    4 references
    private float YForce;
    1 reference
    public float gravity;
    1 reference
    public float jumpForce;
    1 reference
    public float Speed;
    // Start is called before the first frame update
    0 references
    void Start()
    {
        rigid = GetComponent<Rigidbody2D>();
        landSc = GetComponentInChildren<LandScript>();
    }

    // Update is called once per frame
    0 references
    void Update()
    {
        XForce = Speed * Input.GetAxis("Horizontal");
        if(landSc.IsLand){
            if(Input.GetKey(KeyCode.UpArrow)){
                YForce = jumpForce;
            }else{
                YForce = 0;
            }
        }else{
            YForce -= gravity;
        }
        rigid.velocity = new Vector2(XForce,YForce);
    }
}
```

③ インスペクターから jumpForce を 20、Gravity を 1 くらいにしてください。すると↑キーでジャンプができます。

8．カメラ制御

基本的な移動ができるようになったので、ステージの作成に移ります。

① 足場となっているオブジェクト Ground をコピーアンドペーストして、以下のような位置に移動させます。適当で結構です。



シーンビューでの視点の移動はマウスホイール押し込み＋ドラッグまたは矢印キーでできます。拡大縮小はマウスホイールを回せばできます。

② このまま実行すると、プレイヤーが画面外へ出られてしまいます。これを回避するために、視点を動かしましょう。

③ ゲーム実行中の視点は Main Camera（正確にいうと Camera コンポーネントのついたオブジェクト）の位置です。Main Camera に CameraScript という名前でスクリプトを追加しましょう。

```
5   public class CameraScript : MonoBehaviour
6   {
7
        2 references
8       private GameObject Player;
        5 references
9       private float difX;
        4 references
10      private Transform trans;
11      // Start is called before the first frame update
        0 references
12      void Start()
13      {
14          Player = GameObject.FindGameObjectWithTag("Player");
15          trans = GetComponent<Transform>();
16      }
17
18      // Update is called once per frame
        0 references
19      void Update()
20      {
21          difX = Player.transform.position.x - trans.position.x;
22
23          if(difX > 3){
24              trans.Translate(difX - 3,0,0);
25          }
26          if(difX < -3){
27              trans.Translate(difX + 3,0,0);
28          }
29      }
30  }
```

GameObject.FindGameObjectWithTag(“Player”)：Player タグがついたオブジェクトを一つ取得する命令。FindGameObjectsWithTag(“~”)との違いは、一つ取得

するか複数取得（して配列で返す）かです。
自身の X 座標が Player の座標と 3 以上離れたら Player についていくように移動するスクリプトです。

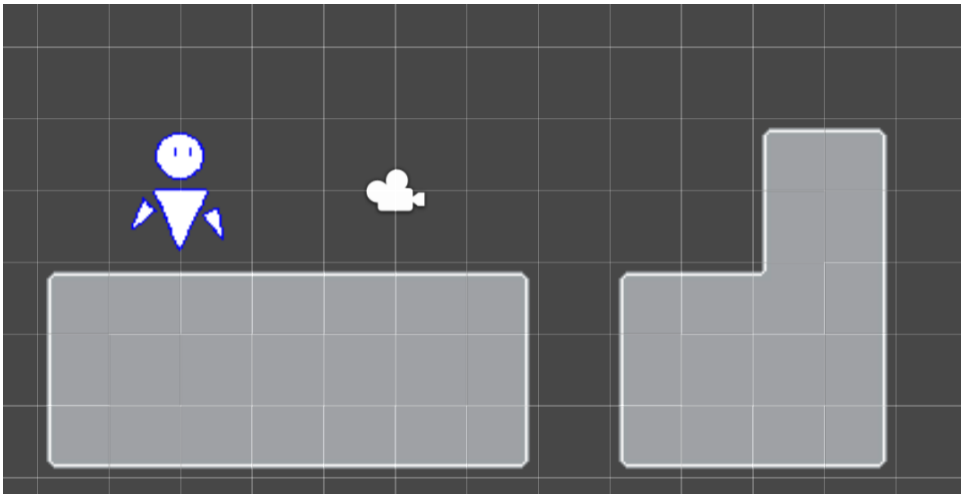
④ 実行すると常にプレイヤーが画面に映り続けます。

9．スプライトの分割

カメラの準備も終わったので本格的にステージを作っていきましょう。

① ８.①みたいにコピペやらなんやらで足場とか壁とかギミックを作るのもいいですが、もっといい方法があるのでそれを教えます。まずは最終的にやりたいことを見せます。

こういう絵から

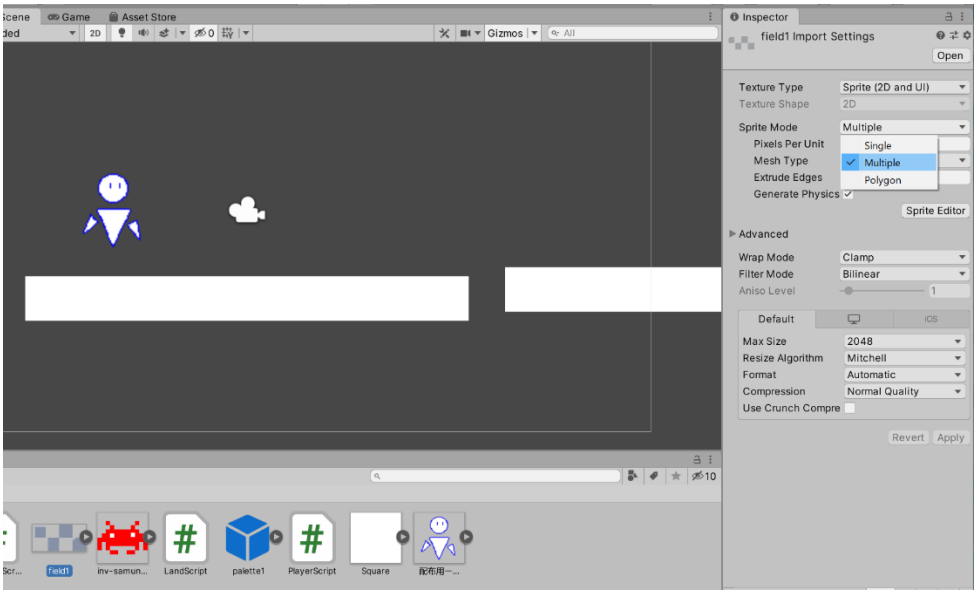


こんなステージを作ります。
絵をパーツとして、いくつかのパーツを組み合わせてステージをつくるつもりです。

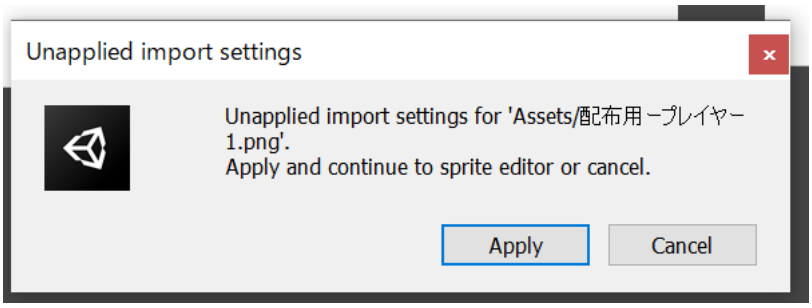
② まずは絵を描きましょう。↑のように、・サイズと形が同じ・つなげたときにきれいにつながる絵を・複数枚一つの絵に・それぞれが接しないように・きれいに配置して描いてください。（うまく説明できてる気がしない、わからなかったら呼んで）描くのがだるかったら呼んでくれたらデータあげます。

③ 描いた絵を取り込みます。

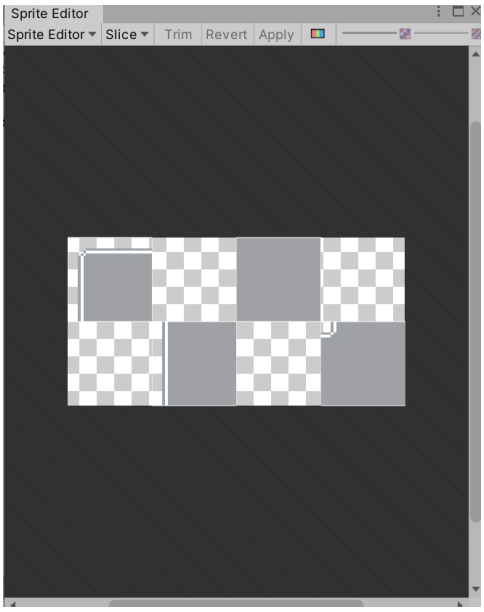
④ 取り込んだ絵をクリックして、インスペクターから Sprite Mode を Single から Multiple にします。こうすると一枚の絵を複数のスプライトとして扱うことができます。



⑤ Sprite Mode のちょっと下にある Sprite Editor ボタンを押してください。この時下のようなダイアログが出たら Apply を押してください。

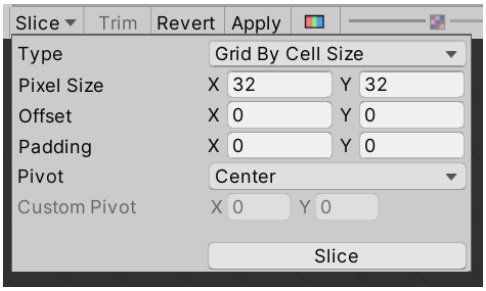


すると下のようなウィンドウが出ます。



この画面で絵の分割を行います。

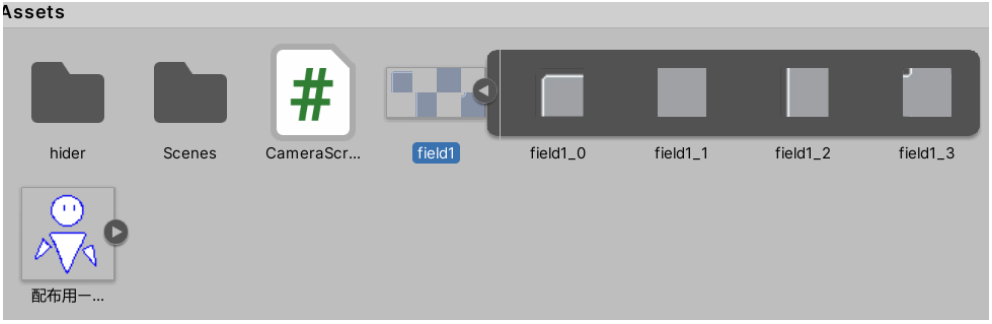
⑥ 絵がきれいに配置されている人は、右上の Slice ボタンから Type に Automatic を選択、または Slice by Cell Size を選択したうえで一パーツの大きさを指定して、最後に Slice ボタンを押してください。



そうでない人は、絵の 1 パーツにしたい部分の上でドラッグをすることを繰り返してください。

⑦ その画面を閉じてください。すると先ほどと同じようなダイアログが出るので、Apply を押してください。

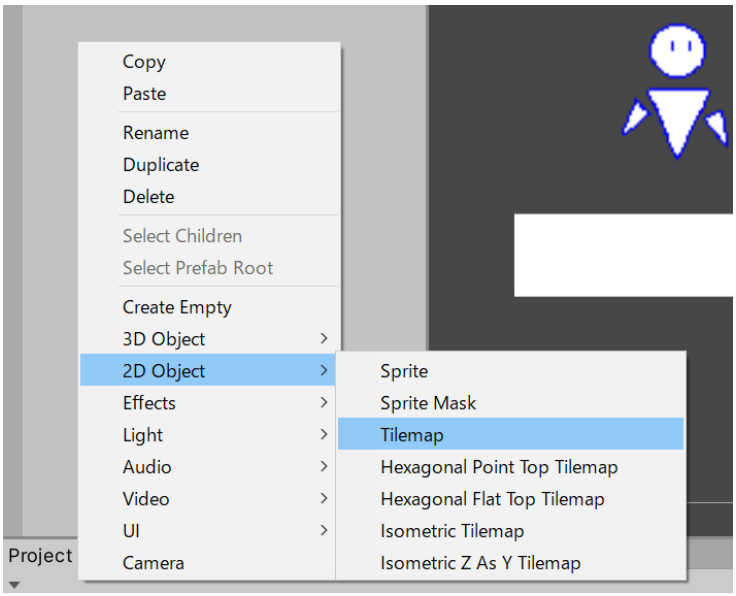
⑧ もとの画面に戻って絵をクリックすると、分割されたデータが出てきます。これらのデータは一つ一つをスプライトとして扱えます。



１０．　Tilemap の準備

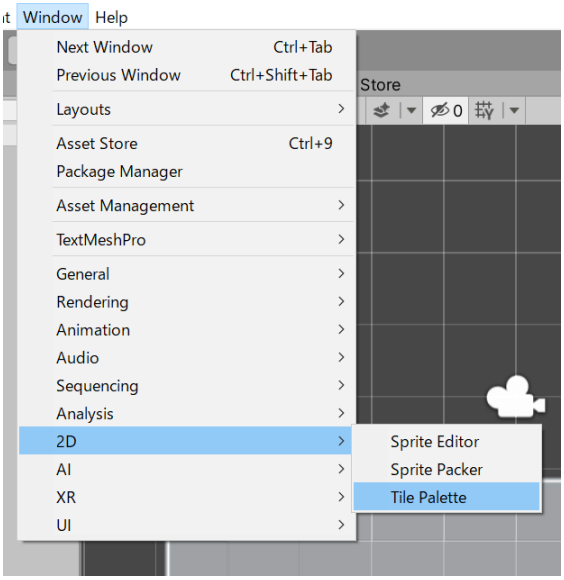
① 用意したスプライトを使っていよいよステージを作ります。これから使うのはタイルマップという機能です。この機能は、スプライトを「タイル」とみなし、シーン上に敷き詰めることでステージの見た目をつくる機能です。

② まずはヒエラルキー上に 2D Object→Tilemap からオブジェクトをつくります。Grid という名前です。



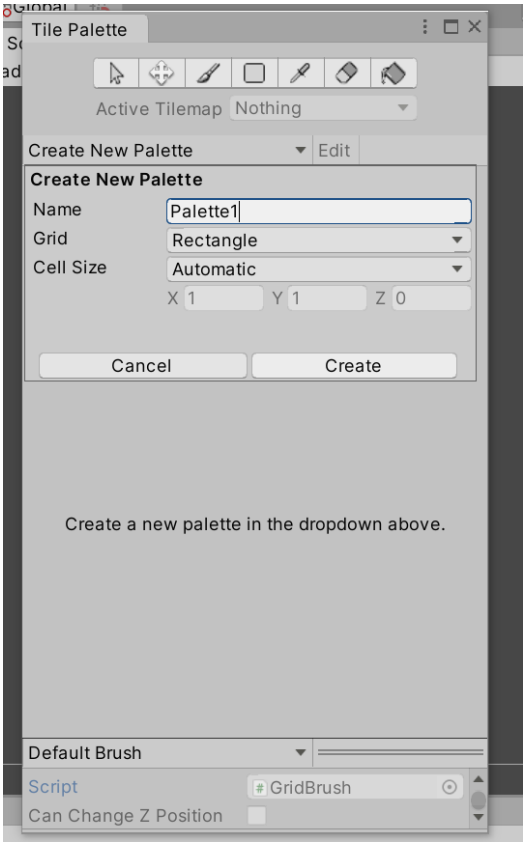
③ これからの作業に邪魔なので、Ground は消してください。

④ 画面上の Window→2D→Tile Palette を押してください。

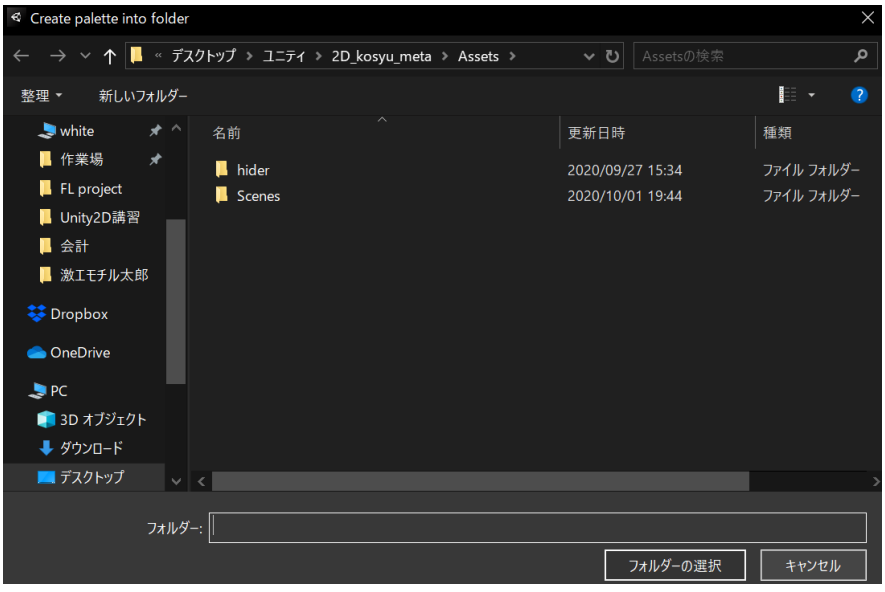


すると新しいウィンドウが出てきます。

⑤ Create new palette を選択して名前を Palette1 にして create を押してください。

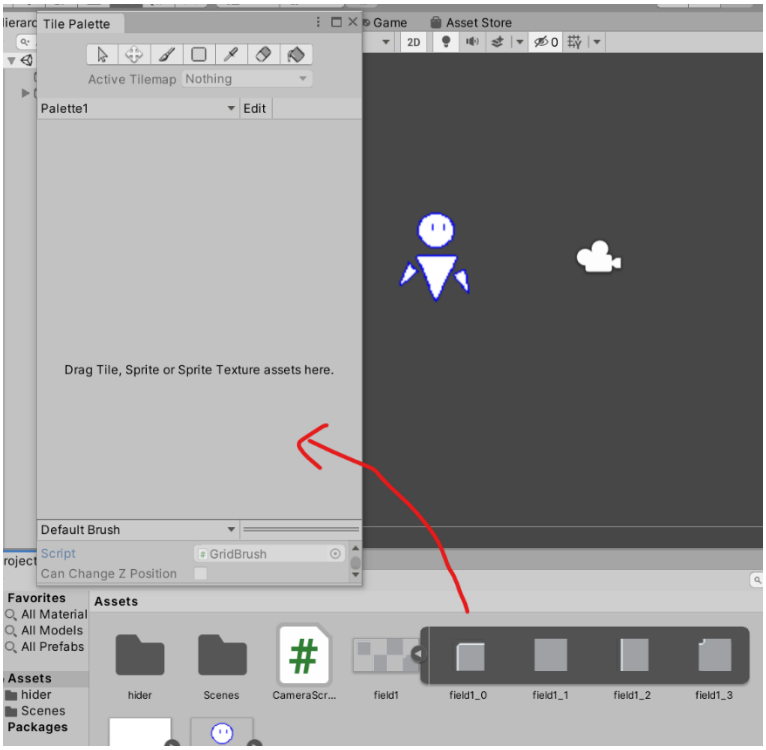


セーブ画面が出たらフォルダーの選択ボタンを押してください。

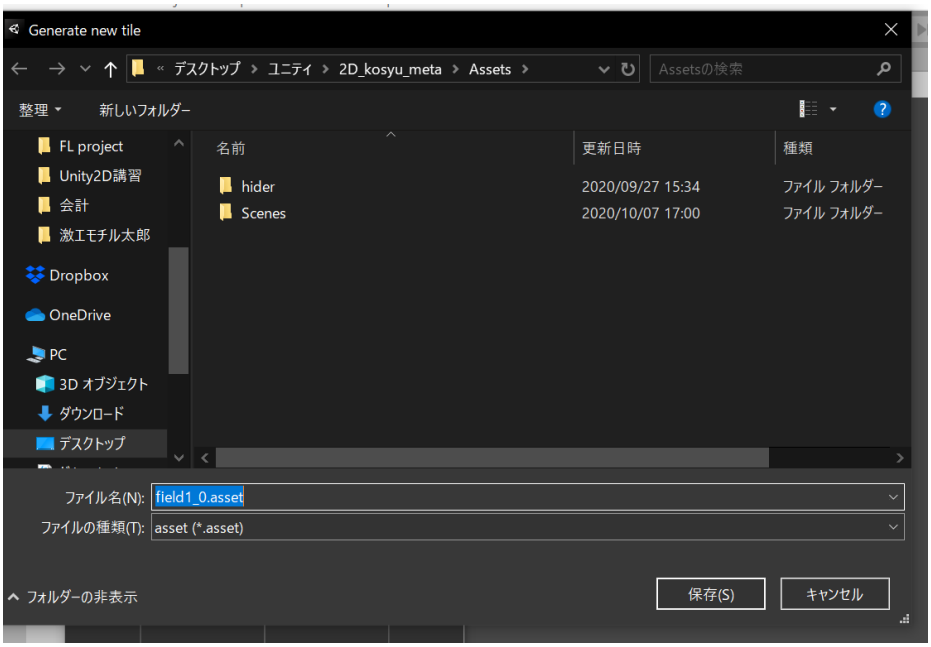


このウィンドウをタイルパレットといい、ここにタイルを登録します。

⑥ タイルパレットに、先ほど分割したスプライトをドラッグアンドドロップします。

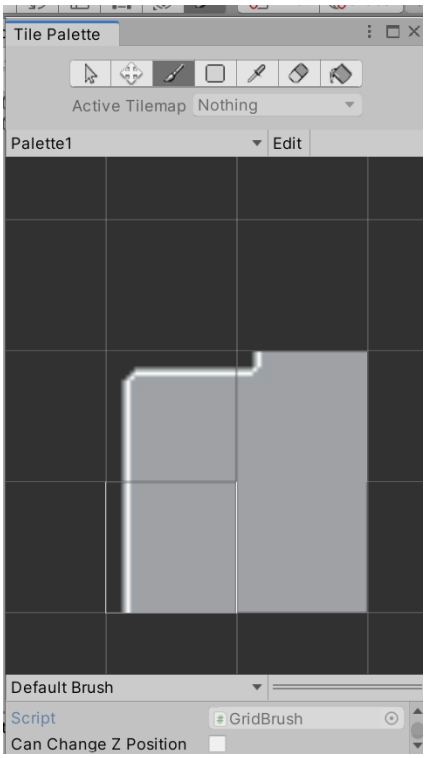


すると保存画面が出るので、そのまま保存してください。



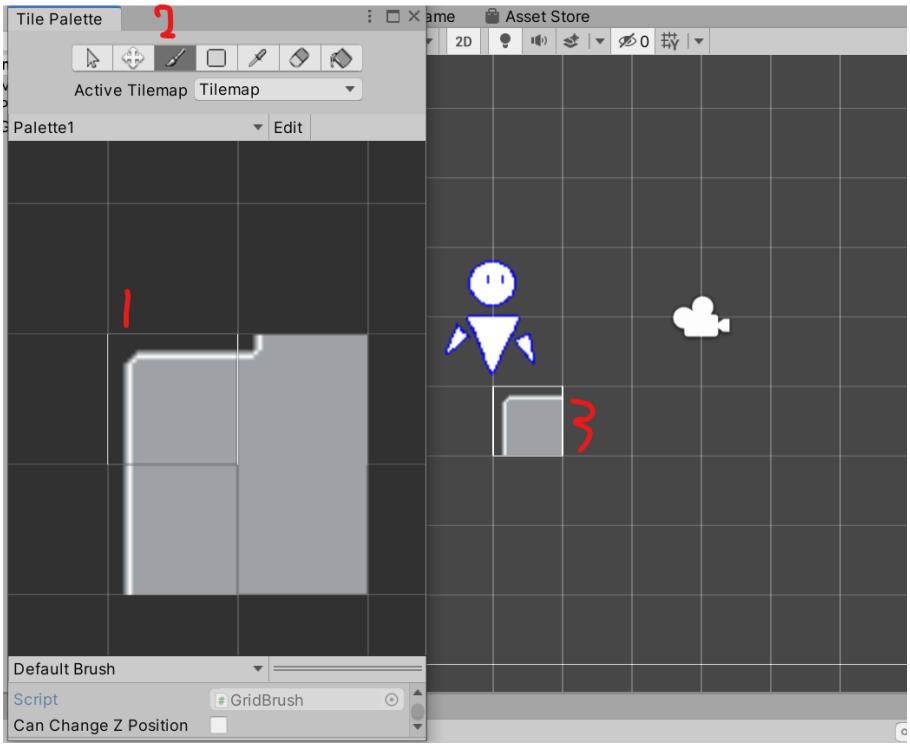
これで、スプライトをタイルという形にして、パレットに登録したことになります。

⑦ ほかの分割したスプライトについても同様にドラッグアンドドロップして保存してください。

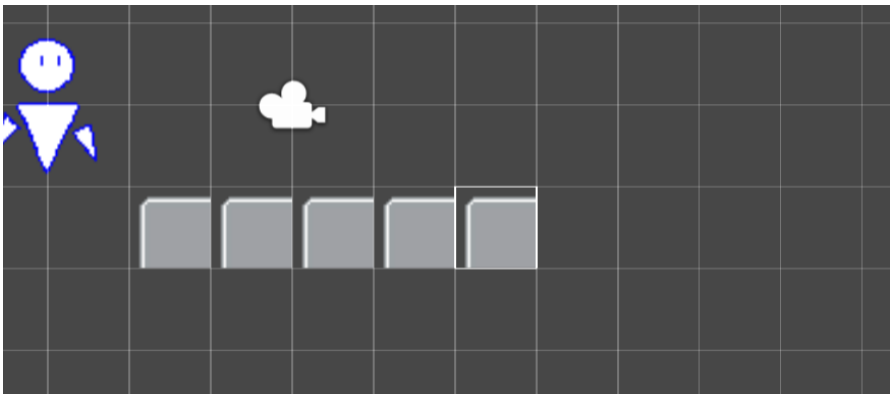


１１．　Tilemap

① パレットの用意ができたので、ステージを作ります。・パレット上から一つタイルを選んでクリックし、・パレット上部のペンのマークが黒くなっていることを確認し、・シーン上にカーソルをもっていってください。

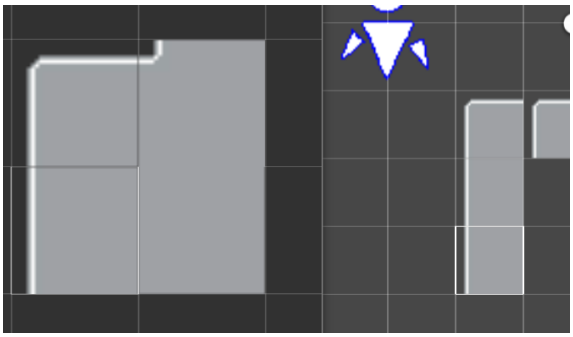


② クリックするとその場にタイルが置かれます。ドラッグすると通った範囲をすべてそのタイルで塗ることが可能です。

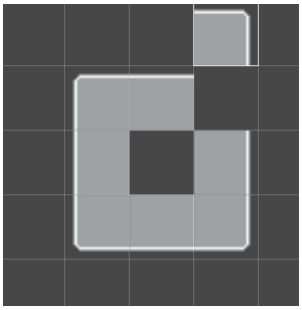


このように、ペイントツール感覚でステージを作ることができます。
以下、使い方を説明していきます。(大きさが違う人は⑦から見て)

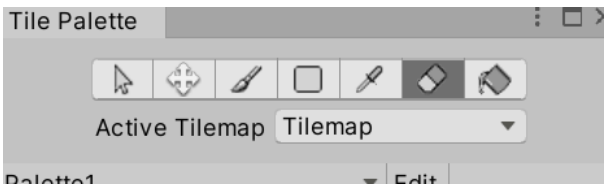
③ 塗るタイルの種類を変えたいときはパレットから別のタイルを選択します。



④ タイルを回転させたいときは、『「』ボタンか『」』ボタンを押してください。



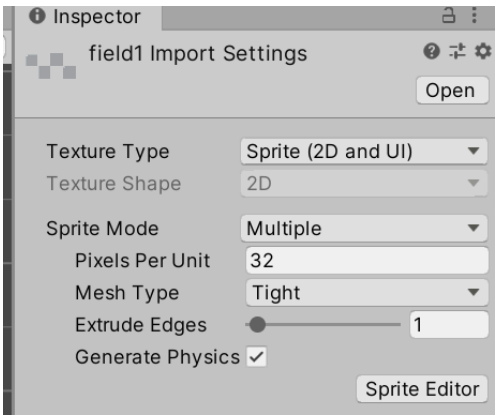
⑤ タイルを消したいときはパレット上部の消しゴムのボタンを押してください。



⑥ では自由にステージをつくってみてください。

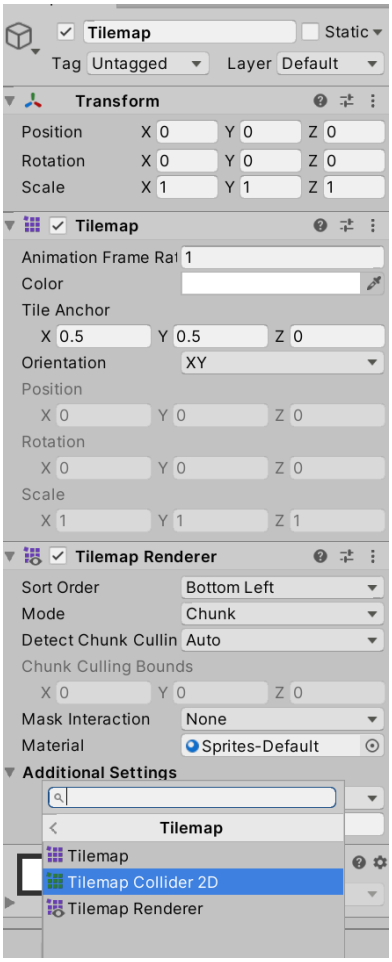
⑦ サイズが合わないという人は、画像データの Pixels per unit を変更してみてくだ

さい。



１２．　当たり判定の追加

① Tilemap で作ったステージには当たり判定がないので、当たり判定を作ります。Grid の子オブジェクトに Tilemap という名前のオブジェクトがあるので、それに Add Component>Tilemap>Tilemap Collider 2D で当たり判定を追加します。

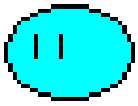


② Tag を ground にすれば当たり判定の完成です。

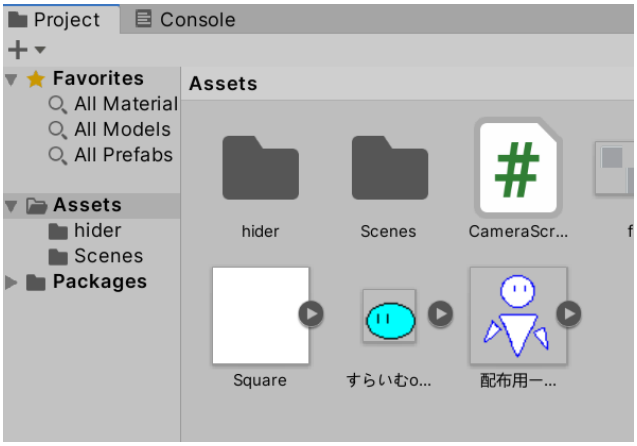
１３．　敵を作る

① ステージができたので敵をつくります。

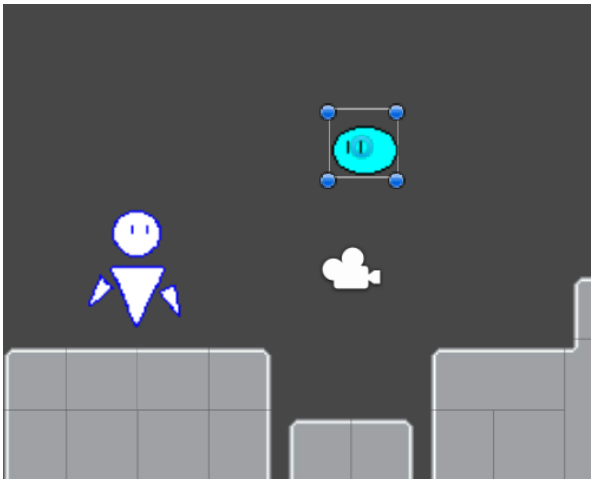
② まずは絵を用意してください。一枚でいいです。



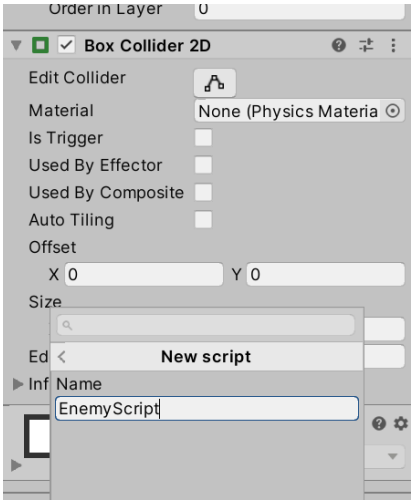
③ プロジェクトに放り込みます。



④ プロジェクトからシーンに放り込みます。



⑤ Box Collider 2D と New Script から EnemyScript を Add Component します。



⑥ 敵の仕様は様々考えられると思いますが、とりあえず当たったら HP が減ることにしましょう。まず自分に HP をつけます。Player Script に以下のように加筆してください。

```
public class PlayerScript2 : MonoBehaviour
{
    private Rigidbody2D rigid;
    private LandScript landSc;
    private float XForce;
    private float YForce;
    public float gravity;
    public float jumpForce;
    public float Speed;
    public int HP = 10;
    // Start is called before the first frame update
    void Start()
    {
        rigid = GetComponent<Rigidbody2D>();
        landSc = GetComponentInChildren<LandScript>();
```

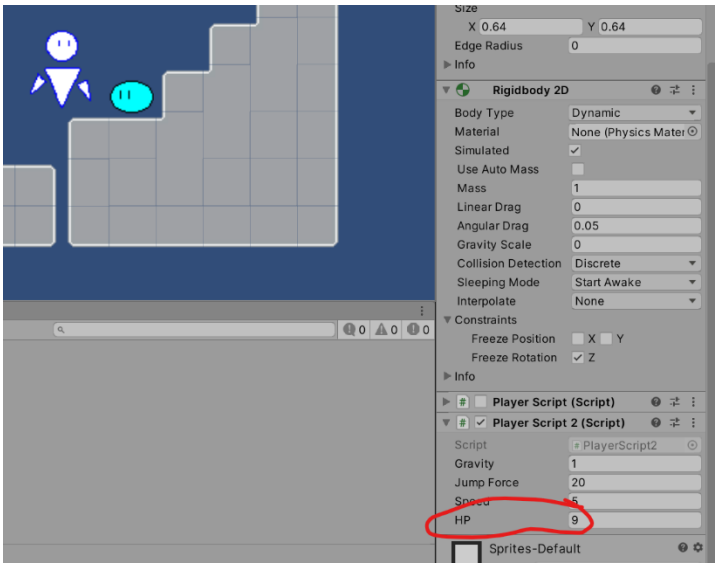
変数「HP」を追加しました。

⑦ 次に、敵と当たったら HP を減らすようにします。この機能は、PlayerScript 中に OnCollitionEnter2D をつけてタグが Enemy だったら……という感じに実装することもできますが、今回は EnemyScript から HP を減らすようにしましょう。EnemyScript を編集します。

```
public class EnemyScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }

    void OnCollisionEnter2D(Collision2D Hit){
        if(Hit.gameObject.CompareTag("Player")){
            Hit.gameObject.GetComponent<PlayerScript>().HP -= 1;
        }
    }
}
```

敵の当たり判定は IsTrigger ではないので、OnTriggerEnter２D ではなく OnCollisionEnter２D になっています。Collision だと Hit.CompareTag ができないので gameObject の情報の取得を間にかませていることに気をつけてください。

⑧ 当たると HP が減るようになりました。



敵の機能はほかにもたくさん考えられると思うので（動くなど)、自分でつけてみてください。(分からなかったら聞いてね)

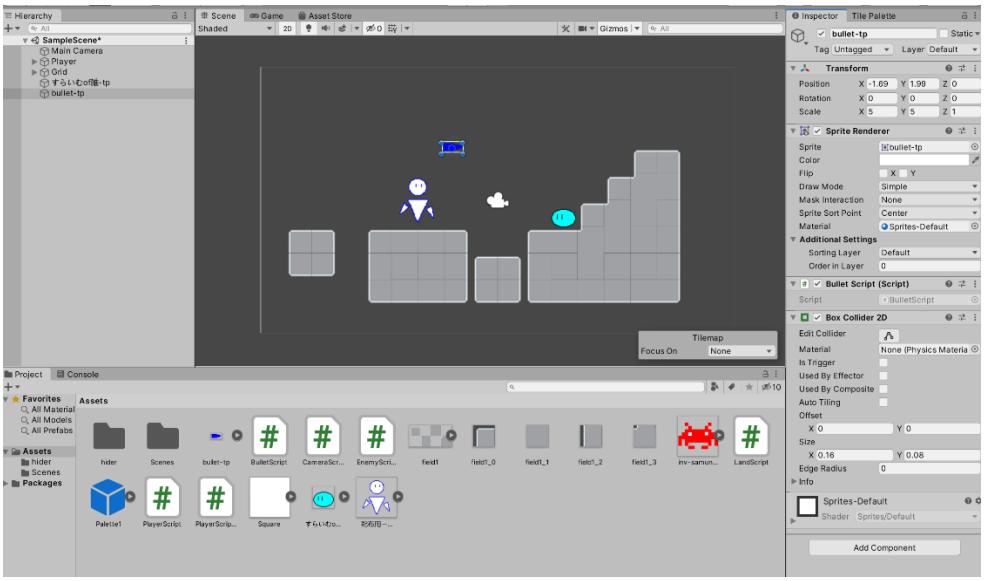
１４． 弾をつくる

① 敵を倒すために弾を発射できるようにしましょう。まずは絵をかきます。

② 敵の時と同じ要領でプロジェクトとシーンに放り込みます。

③ 敵のときと同じく BoxCollider2D と New Script から BulletScript を追加します。

BoxCollider2D は IsTrigger を On にしてください。

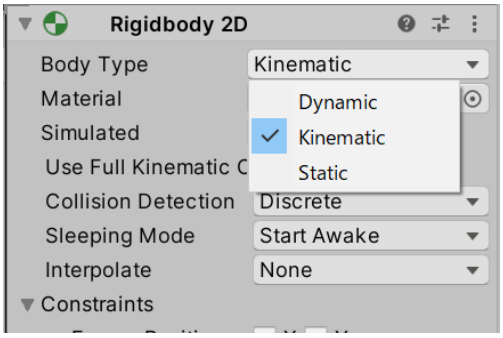


④ 弾は自動で動くようにしましょう。Bulletscript を編集します。

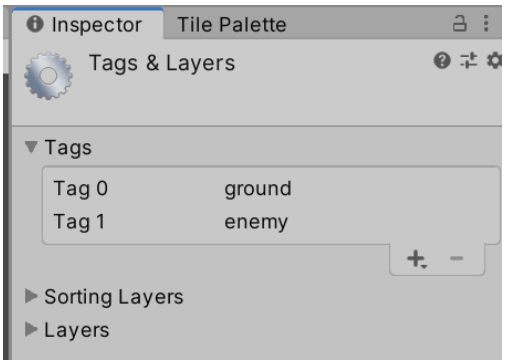
```
5   public class BulletScript : MonoBehaviour
6   {
7       // Start is called before the first frame update
        0 references
8       void Start()
9       {
10
11      }
12
13      // Update is called once per frame
        0 references
14      void Update()
15      {
16          GetComponent<Transform>().Translate(0.1f,0,0);
17      }
18
```

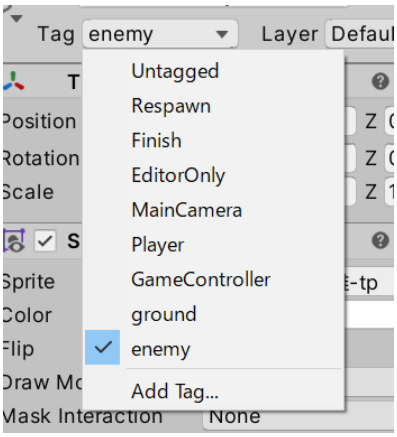
Transform.Translate(x,y,z)：オブジェクトの位置を x,y,z だけ動かす命令。GetComponent<Transform>()で自分自身の Transform コンポーネントを取得しているので、自身の位置を毎フレーム x 方向に 0.1 だけ動かす、という意味になります。

⑤ 弾が敵に当たったら敵が消えるようにしましょう。当たり判定を検知するために弾に rigidbody2D をつけ、Body Mode を kinematic にしてください。



⑥ 次に、当たったのが敵かどうかを判断するため、敵に Tag をつけます。敵オブジェクトのタグを Add Tag から enemy にしてください。





⑦ 次に BulletScript を編集します。

```
    void Update()
    {
        GetComponent<Transform>().Translate(0.05f,0,0);
    }

    0 references
    void OnTriggerEnter2D(Collider2D Hit){
        if(Hit.CompareTag("enemy")){
            Destroy(Hit.gameObject);
            Destroy(gameObject);
        }
    }
}
```

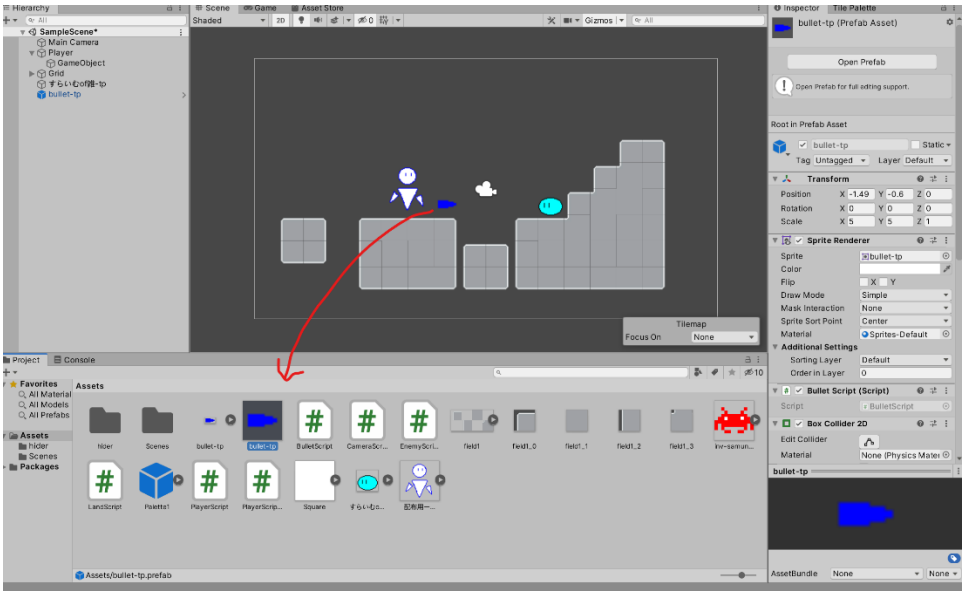
Destroy(Hit.gameObject)は当たったオブジェクト、Destroy(gameObject)は自分自身を消す命令です。

⑧ 実行すると、弾が敵にあたると弾も敵も消えることがわかります。

１５．　弾の発射

この弾をプレイヤーが発射できるようにします。

① シーン上の弾をプロジェクトビューにドラッグアンドドロップし、プレハブ化します。



② もとにした弾は削除してください。

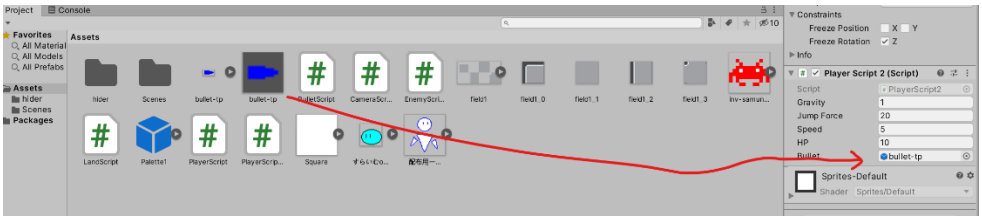
③ PlayerScript を編集します。

```
14      public int HP = 10;
15      public GameObject bullet;
16      // Start is called before the first frame update
17      void Start()
18      {
19          rigid = GetComponent<Rigidbody2D>();
20          landSc = GetComponentInChildren<LandScript>();
21      }
22
23      // Update is called once per frame
24      void Update()
25      {
26          XForce = Speed * Input.GetAxis("Horizontal");
27          if(landSc.IsLand){
28              if(Input.GetKey(KeyCode.UpArrow)){
29                  YForce = jumpForce;
30              }else{
31                  YForce = 0;
32              }
33          }else{
34              YForce -= gravity;
35          }
36          rigid.velocity = new Vector2(XForce,YForce);
37
38          if(Input.GetKeyDown(KeyCode.Space)){
39              Instantiate(bullet,transform.position,Quaternion.Euler(0,0,0));
40          }
41      }
```

変数 bullet を追加し、スペースキーが押されたら弾が発射されるようにしました。

if(Input.GetKeyDown())：そのキーが押されたときだけかっこ内の命令を実行する。GetKey()との違いは、GetKey は押されている間ずっと、だが、GetKeyDown は押され始めたときのみ。

④ インスペクターから PlayerScript の bullet のところに先ほどのプレハブをドラッグアンドドロップします。



⑤ 実行すると、スペースキーが押されるたびに弾が発射されます。

１６．　弾の削除

① このままだと弾は敵に当たらない限り消えないため、弾を何発も撃つと処理落ちするようになります。そのため、弾に射程を持たせ、射程を越えたら弾が消えるようにします。BulletScript を編集します。

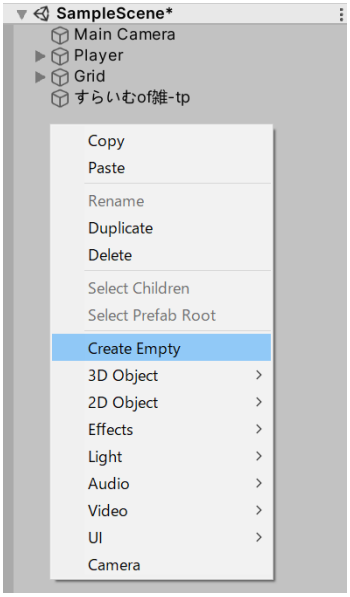
```
14      void Update()
15      {
16          GetComponent<Transform>().Translate(0.05f,0,0);
17          if(transform.position.x > 10){
18              Destroy(gameObject);
19          }
20          if(transform.position.x < -5){
21              Destroy(gameObject);
22          }
23      }
24
```

弾の x 座標が１０以上になるかー５以下になった時に弾が消えるようにしました。ステージの作り方によっては、x 座標が１０以上になっても弾には消えないでほしい、ということもあると思います。そういう時はここの数値を変えるなり別の基準（弾が発射されてからの秒数など）で弾を消すなりしてください。

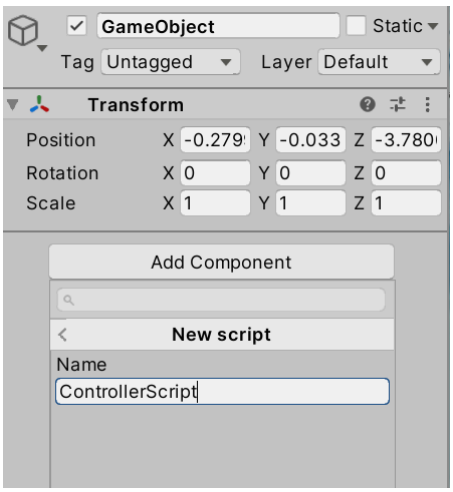
１７．　ゲームオーバーの実装

先ほどプレイヤーに HP をつけましたが、このままでは HP が 0 になっても何も起こらず、マイナスになることもあります。そこで HP が 0 になったとき、ゲームオーバーになるようにしましょう。

① ゲーム進行にかかわる処理をやってもらうオブジェクトを作りましょう。Create Empty で空のオブジェクトを作ります。



② New Script から ControllerScript を追加して、以下のように編集します。




```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5
    0 references
6   public class ControllerScript : MonoBehaviour
7   {
8       // Start is called before the first frame update
        2 references
9       private GameObject Player;
        2 references
10      private PlayerScript ps;
        0 references
11      void Start()
12      {
13          Player = GameObject.FindGameObjectWithTag("Player");
14          ps = Player.GetComponent<PlayerScript>();
15      }
16
17      // Update is called once per frame
        0 references
18      void Update()
19      {
20          if(ps.HP <= 0){
21              SceneManager.LoadScene(SceneManager.GetActiveScene().name);
22          }
23      }
24  }
25
```

４行目も追加していることに注意してください。

SceneManager.GetActiveScene().name：今実行されているシーンの名前を取得します。SceneManager.LoadScene と組み合わせて、現在のシーンを再読み込みし、初めからやり直すことができます。

③ 実行すると、敵に１０回当たると初めからやり直しになりました。ゲームオーバーはほかの実装法も思いつくと思うのでぜひ他の方法でも試してください。

第一回 2D 講習はこれで終わりです。