【Step2】01_ステージを作ろう 小テスト

1. **次の説明が正しい場合は「真」、間違っている場合は「偽」を選びなさい。**

タイルパレットウィンドウを出すには、上部のメニューから ウィンドウ > 2D> タイルパレット を選択する。

- (●) 真
- ( ) 偽

2. **次の説明が正しい場合は「真」、間違っている場合は「偽」を選びなさい。**

タイルマップを作成するには、プロジェクトウィンドウの「＋」をクリックして、2Dオブジェクト> タイルマップ を選択する。

- ( ) 真
- (●) 偽

タイルマップを作成するには、ヒエラルキーウィンドウの「＋」をクリックして、2D オブジェクト> タイルマップ を選択します。

3. **タイルパレットウィンドウを出した後、タイルパレットウィンドウにタイルをドラッグ＆ドロップしましたがパレットにタイルが追加されませんでした。理由として考えられるものを選びなさい。**

- ( ) タイルマップが作成されていないから
- (●) パレットが作成されていないから
- ( ) ドラッグしたタイルがスライスされていなかったから
- ( ) Assetsフォルダの中に新しいフォルダを作っていないから

パレットにタイル（絵具）を追加するためには、まずパレットを作成する必要があります。「新しいパレットを作成」をクリックし、名前を付け、新しいフォルダを作ってその中に保存しましょう。「パレットを新しく作る」のと、「パレットにタイルを新しく追加する」のは似ていて分かりにくいですが、しっかり区別して覚えましょう！タイルはスライスされていなくてもパレットに追加することは出来ますが、スライスされていなければ、レッスンでやったように細かくステージを作っていくことはできません。

【Step2】02_プレイヤーに重力・当たり判定をつけよう 小テスト

1. **重力をかけるなどの、物理的な動きをシミュレートしたいときは、下のどのコンポーネントを追加すればよいでしょう。適切なものを選びなさい。**

- ○ スプライトレンダラー(Sprite Renderer)
- ◉ リジッドボディ(Rigidbody)
- ○ ボックスコライダー(Box Collider)

スプライトレンダラー(Sprite Renderer)は画像を表示するコンポーネントです。

2. **プレイヤーにRigidbody 2DとBox Collider 2Dを付けましたが、地面をすり抜けてしまいます。 その理由として正しいものを選びなさい。**

- ◉ 地面にコライダーが付いていない
- ○ 地面にリジッドボディが付いていない
- ○ プレイヤーにPlayerControllerスクリプトが付いていない

プレイヤーと地面の両方にコライダーを付けないとすり抜けてしまいます。

3. **プレイヤーがステージの端に進んだ時、回転して落ちてしまいました。 落ちていく時、回転しないようにする方法として、正しいものを選びなさい。**

- ○ プレイヤーのRigidbody 2DのConstraintsの位置を固定Xにチェックを入れる
- ○ プレイヤーのRigidbody 2DのConstraintsの位置を固定Yにチェックを入れる
- ◉ プレイヤーのRigidbody 2DのConstraintsの回転を固定Zにチェックを入れる

Rigidbody 2D の Constraints の回転を固定 Z にチェックを入れると、ゲームオブジェクトが回転しなくなります。 位置を固定にチェックをすれば、X なら横、Y なら縦に動かなくなりますが、Translate メソッドなどで座標を書き換えている場合は普通に動きます。 そのことからも、リジッドボディを使っている時に座標を直接書き換えるのが、あまりよくないことが分かりますね。

【Step2】03_プレイヤーをステージ上で動かそう 1 小テスト

1. **Velocity（速度）についての説明で正しいものを全て選びなさい。**

- [ ] VelocityのYが-3の時、左に進む。
- [x] Velocityを変更したい場合、2Dのゲーム作成時はRigidbody2Dコンポーネントをアタッチする必要がある。
- [ ] 速度とは速さのことである。
- [x] VelocityのXが3の時、右に進む。

速度は、速さ＋向きの情報を持っています。X の速度が-5 なら、5 の速さで左に移動します。

2. **正しいものを選びなさい。**

キーボードの右を押した時にPlayerを右に移動させたい場合、 PlayerControllerスクリプトの、if (Input.GetKey(“right”)){ }の中にどの命令を入れればよいでしょうか。正しいものを選びなさい。
※レッスンと同様にコンポーネント等がアタッチされていると仮定する。

- (●) this.GetComponent<Rigidbody2D>().velocity = new Vector2(3,0);
- ( ) this.GetComponent<Rigidbody>().velocity = new Vector2(3,0);
- ( ) this.GetComponent<Rigidbody2D>().velocity.x = 3;
- ( ) this.GetComponent<Rigidbody2D>().velocity = Vector2(3,0);
- ( ) this.Rigidbody2D.velocity = new Vector2(3,0);

Transform 以外のコンポーネントを使いたい場合、GetComponent を使う必要があります。 レッスンでは Player に Rigidbody2D を付けていたので Rigidbody コンポーネントではなく、Rigidbody2D コンポーネントを取得する必要があります。 速度を変更したい場合は、「.velocity」として、そこに例えば「new Vector2(3,0)」等を代入します。

3. **Playerを移動させた時、Playerのコライダーが Tilemapのコライダーに引っかかってたまに止まってしまうバグが出てしまいました。 それを修正するために、Composite Collider 2D（2D複合コライダー）をTilemapにアタッチしましたが、もう少し設定が必要です。 その設定として、正しいものを選びなさい。**

- (●) TilemapのRigidbody2Dのボディタイプを「静的」にする。
- ( ) PlayerのRigidbody2Dのボディタイプを「静的」にする。
- ( ) Composite Collider 2Dの「コンポジットで使用」にチェックを入れる。
- ( ) Playerのコライダーの「コンポジットで使用」にチェックを入れる。

Composite Collider 2D（2D 複合コライダー）を Tilemap にアタッチした時に、自動で Rigidbody2D もアタッチされます。 初期状態では、Tilemap の Rigidbody2D のボディタイプは「動的」なので、再生した時に Tilemap（ステージ）が下に落ちていきましたね。 それを防ぐために、Tilemap の Rigidbody2D のボディタイプは「静的」にしてください。字の通り、静的にすると動かなくなり、動的にすると重力などの影響を受けると動くようになります。 また、「コンポジットで使用」にチェックを入れると、コライダー同士がくっついて 1 つになります。 1 つのコライダーとして扱いたいのは、Tilemap Collider 2D なので、Tilemap Collider 2D の「コンポジットで使用」にチェックを入れる必要があります。

【Step2】04_プレイヤーをステージ上で動かそう 2 小テスト

1. **2D物理マテリアルについての説明で、正しいものを全て選びなさい。 ※物理マテリアル等は正しく設定されているとする。**

- [ ] 2D物理マテリアルの「Bounciness」は摩擦という意味で、値を小さくすればすべりやすくなる。
- [x] 2D物理マテリアルを新規作成するには、プロジェクトウィンドウで「+」を押してから、2D物理マテリアルを選択する
- [ ] 2D物理マテリアルを新規作成するには、ヒエラルキーウィンドウで「+」を押してから、2D物理マテリアルを選択する。
- [ ] 2D物理マテリアルの「Friction」を大きくすればするほど、高い位置まで跳ね返るようになる。
- [x] 2D物理マテリアルの「Bounciness」を「0」にすると、跳ね返らなくなる。
- [ ] 2D物理マテリアルの「Friction」は摩擦という意味で、値を大きくすればすべりやすくなる。

「Friction」と「Bounciness」は Unity エディタで日本語化されなかったので、英語のまま出題しました。 「Friction」は摩擦という意味で、この値を大きくすればすべりにくくなります。逆に、「0」に近づけるほどすべりやすくなります。 また「Bounciness」は弾力性という意味で、この値を「1」にすれば元の位置まで跳ね返り、「0」にすると跳ね返らなくなります。

2. **PlayerControllerスクリプトのStart関数に以下のコードを書きました。**

```
if (true)
{
    Debug.Log("1");
}
else if (true)
{
    Debug.Log("2");
}
else
{
    Debug.Log("3");
}
```

再生した時にコンソールに表示される数字を選びなさい。

- (●) 1
- ( ) 2
- ( ) 3

if～else をーつのかたまりととらえてください。スクラッチでやった黄色いパーツのかたまりと同じですね。 1 つ目の if の条件を満たしている（true）ので、if の中の命令が実行され、コンソールにはまず「1」が表示されます。 その下の else は、「でなければ」という意味なので、上の条件を満たしている以上それ以降は考える必要がないですね。 よって、その時点でかたまりを抜けるので、コンソールには「1」しか表示されません。

3. **PlayerControllerスクリプトのStart関数に以下のコードを書きました。**

```
if (false)
{
    Debug.Log("1");
}
else
{
    Debug.Log("2");
}

if (false)
{
    Debug.Log("3");
}
else if (true)
{
    Debug.Log("4");
}
else
{
    Debug.Log("5");
}
```

再生した時にコンソールに表示される数字を全て選びなさい。

- [ ] 1
- [x] 2
- [ ] 3
- [x] 4
- [ ] 5

今回はかたまりが２つあります。 まず、上のかたまりを見てください。 今回は if の条件を満たさない（false）ので、その下の else の中の命令が実行され、「2」が表示されます。 下のかたまりでも、if の条件を満たさないので、その下の else if の条件をチェックします。else if の条件は true なので、命令が実行され「4」が表示されます。 それ以降の else は、問 1 と同様に考える必要がないので、かたまりを抜けます。 今回は分かりやすいように、上のかたまりと下のかたまりを改行で分けましたが、改行がなくても結果は同じです。

【Step2】05_変数を使ってみよう 1 小テスト

1. 下の画像のプログラムを実行すると、ねこはなんと言うでしょうか。 正しいものを選びなさい。

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5
- ○ 6
- ○ 7
- ○ 8
- ◉ 9
- ○ 10
- ○ 0

変数は箱みたいなもので、中身を保持して（=覚えておいて）くれます。 例えば、「変数を 2 にする」という命令は、変数という箱の中身を「2」に変えます。 「変数を 2 ずつ変える」という命令は、変数という箱の中身に「2」を足します。 この 2 つは似ているけれど違います。分かりにくければ、動画をもう一度見て復習しておきましょう。

2. **PlayerControllerスクリプトを以下のように変更しました。**

```
using UnityEngine;
public class PlayerController : MonoBehaviour
{
    int vx;//プレイヤーのxの速度

    //スタート関数が実行された後、繰り返し実行される
    void Update()
    {
        if (Input.GetKey("right"))//右が押されている時
        {
            vx = -3;
            this.GetComponent<Rigidbody2D>().velocity = new Vector2(vx, this.GetComponent<Rigidbody2D>().velocity.y);
        }
        else if (Input.GetKey("left"))//左が押されている時
        {
            vx = 3;
            this.GetComponent<Rigidbody2D>().velocity = new Vector2(0, this.GetComponent<Rigidbody2D>().velocity.y);
        }
        else//右も左も押されていない時
        {
            vx = 0;
            this.GetComponent<Rigidbody2D>().velocity = new Vector2(vx, this.GetComponent<Rigidbody2D>().velocity.y);
        }
    }
}
```

ゲームを再生した時の説明として、正しいものを全て選びなさい。
※動画と同じようにコンポーネント等は設定されているものとする

- [ ] キーボードの左を押した時、Playerは右に進む
- [x] キーボードの左を押した時、vxの中身は「3」になる
- [x] キーボードの右を押した時、Playerは左に進む
- [ ] キーボードの右を押した時、Playerは右に進む

キーボードの右を押した時は、一番上の if の条件を満たすので、if の中の命令だけが実行されます。 vx = -3;と命令してるので、vx という変数の中に「-3」が入り、次の命令で Player の x の速度を vx(-3)に変更しているので、左に進みます。 キーボードの左を押した時は、真ん中の else if の条件を満たすので、else if の中の命令だけが実行されます。 vx = 3;と命令してるので、vx という変数の中には「3」が入りますが、次の命令で Player の x の速度を直接「0」に変更しているので、右にも左に進みません。

3. **PlayerControllerスクリプトを以下のように変更しました。**

```
using UnityEngine;
public class PlayerController : MonoBehaviour
{
    Rigidbody2D rigidbody2D;

    // ゲームが再生されたときに、一度だけ実行される
    void Start()
    {

    }
}
```

Start関数の中に以下の命令を追加した時、Rigidbody2Dコンポーネントが変数に正しく代入出来るものを選びなさい。
※動画と同じようにコンポーネント等は設定されているものとする

- ○ rigidbody2D = this.GetComponent<Rigidbody2D>;
- ◉ rigidbody2D = this.GetComponent<Rigidbody2D>();
- ○ rigidbody2D = this.GetComponent<rigidbody2D>();
- ○ Rigidbody2D = this.GetComponent<Rigidbody2D>();

取得（ゲット）したいコンポーネントが入る変数を作って、変数 = this.GetComponent＜取得したいコンポーネント＞();とすると、変数にコンポーネントを代入出来ます。＜＞の後の()を忘れないようにしてくださいね！ Visual Studio では入力候補が表示されるので、完璧に覚えていなくてもプログラムを書くことが出来ますが、よく使うものは覚えてしまいましょう。 今回は変数名を「rigidbody2D」にしていますが、変数名を「rb2D」とした場合は、rb2D = this.GetComponent＜Rigidbody2D＞();とすれば正しく代入することが出来ますよ！

【Step2】06_変数を使ってみよう 2 小テスト

1. **PlayerControllerスクリプトを以下のように変更しましたが、エラーが出てしまいました。**

```
using UnityEngine;
public class PlayerController : MonoBehaviour
{
    int moveSpeed = 3.5;
    int vx;//プレイヤーのxの速度

    // ゲームが再生されたときに、一度だけ実行される
    void Start()
    {
        vx = moveSpeed;
    }

    //スタート関数が実行された後、繰り返し実行される
    void Update()
    {

    }
}
```

エラーを消すために必要な変更を、以下から全て選びなさい。

- [ ] Start関数の中の、「vx = moveSpeed;」をUpdate関数の中に移動する
- [x] 「moveSpeed」に代入している「3.5」を「3.5f」にする
- [x] 「int moveSpeed」を「float moveSpeed」にする
- [x] 「int vx」を「float vx」にする

このエラーは「int moveSpeed = 整数;」（例：int moveSpeed = 3;）とするだけで直りますが、 選択肢にはないので、「moveSpeed」に小数を代入出来るようにする方法を答える問題だと分かります。「int moveSpeed」とすると、「moveSpeed」の中には整数（int 型）しか入れられないので、「int」の代わりに「float」をつけて、小数（float 型）が入るようにしてあげましょう。 小数の最後には「f」を付けないとダメでしたね。 Start 関数の中では「vx」に「moveSpeed（小数）」を代入しているので、「vx」にも同じように「float」を付けて、小数が入るようにしましょう。 「vx = moveSpeed;」を Update 関数の中に移動したとしても、エラーの解決にはなりません。

2. **アクセス修飾子に関する説明で正しいものを、以下から全て選びなさい。**

- [x] 変数を作る時に「private」を付けなくても、変数は自動で「private」として扱われる
- [x] 変数を作る時に「public」を付けると、Unityエディタのインスペクターウィンドウに変数が表示されるようになる
- [ ] 変数を作る時に「public」を付けると、他のクラスから変数の中身を変更出来なくなる

「public」は、「公共の」という意味でしたね。公共交通機関からの連想で、「public」を付けると、誰でも使えると覚えましょう！ 「private」を付けた場合は、他のクラスから変数の中身を変更出来なくなります。 何も付けない場合も「private」として扱われるので、他のクラスから変数の中身を変更することは出来ません。

3. **以下のようなスクリプトを作成しました。**

```
using UnityEngine;
public class Test : MonoBehaviour
{
    int a = 1;
    float b = 0.5f;
    float c = 10;

    void Start()
    {

    }
}
```

Start関数に追加した時にエラーが出てしまうコードを、以下から全て選びなさい。

- [x] a = b;
- [x] a = c;
- [ ] b = a;
- [ ] b = c;
- [ ] c = a;
- [ ] c = b;

この問題はかなり複雑です。しっかり考えてくれた人ほど間違いやすいかもしれません。 「int 型」の変数には「float 型」の変数は代入出来ませんでしたね。 そして、動画の最後の方で「float 型」の変数には、「float 型」の変数も「int 型」の変数も両方とも代入することが出来ると説明しました。 この問題の「変数 b」のように、「float 型」で中身が小数のパターンは動画でやりましたが、「変数 c」のように、「float 型」で中身が整数（=10）のパターンは初めてですね。 「int」を付けると整数を入れることが出来ると説明していたので、「a=c;」が出来ると思った人がいるかもしれないですね。「変数 c」を作る時に「float」と付けているので、「変数 c」は中身が整数だろうと「float 型」として扱います。ですので、上にも書いた通り「int 型」の変数に「float 型」の変数を代入することは出来ず、「a=c;」とするとエラーが出てしまいます。

【Step2】07_プレイヤーをジャンプさせよう 小テスト

1. **Input.GetKeyDown(“space”);の説明として正しいものを選びなさい。**

- ○ スペースキーが（押されている状態から）離された時、１度だけ「true」になる
- ○ スペースキーが押されている間はずっと「true」になる
- ◉ スペースキーが押された時、１度だけ「true」になる

スペースキーが押されている間、ずっと「true」になるのは「Input.GetKey」の場合です。 動画ではやりませんでしたが、 スペースキーが（押されている状態から）離された時、１度だけ「true」にしたければ、「Input.GetKeyUp」というものを使います。

2. **Visual Studioのキーボードショートカットで正しいものを全て選びなさい。**

- ☑ 「Ctrl」+「X」：切り取り
- ☑ 「Ctrl」+「V」：貼り付け
- ☐ 「Ctrl」+「S」：コピー
- ☑ 「Ctrl」+「D」：複製

「Ctrl」+「S」は保存のショートカットです。コピーのショートカットは「Ctrl」+「C」なので、間違えないようにしましょう。 ショートカットキーを覚えると、作業スピードが格段に上がるので、どんどん使って慣れていきましょう。

【Step2】08_地面にいるかどうかの判断をしよう 小テスト

1. **「bool isGrounded;」として、変数を作りました。 この時の説明として正しいものを全て選びなさい。**

- [ ] 「isGrounded」の中には、「true」が入っている
- [x] 「bool」の前に「public」を付けると、インスペクターウィンドウで確認することが出来る
- [x] 「isGrounded」の中には、「true」か「false」を代入することが出来る
- [x] 「isGrounded」を「if」の条件として使うことが出来る

bool 型の変数を作った時、何も代入しなければ、中身が「false」になります。 最初に中身を「true」にしたければ、「bool isGrounded = true;」としてください。

2. **以下の選択肢から、関数名とその説明が正しいものを全て選びなさい。**

※呼ばれる＝関数の中身が実行される

- [x] OnCollisionEnter2D：コライダー同士がぶつかった時に、1度だけ呼ばれる
- [ ] OnCollisionHit2D：コライダー同士がぶつかった時に、1度だけ呼ばれる
- [x] OnCollisionExit2D：接触しているコライダー同士が離れた時に、1度だけ呼ばれる
- [ ] OnCollisionOut2D：接触しているコライダー同士が離れた時に、1度だけ呼ばれる
- [ ] OnCollisionStay2D：コライダー同士が接触している時に、1度だけ呼ばれる

OnCollisionStay2D 関数は、コライダー同士が接触している時に呼ばれ続けます。OnCollisionHit2D と OnCollisionOut2D という関数はありません。

3. **以下のスクリプトの「if」の()に入れられるもの（条件式）として、正しいものを全て選びなさい。**

```
using UnityEngine;
public class PlayerController : MonoBehaviour
{
    int a = 3;
    bool b = true;

    void Update()
    {
        if (            )//←この()にどれが入るか
        {
            Debug.Log("こんにちは");
        }
    }
}
```

- [x] Input.GetKey("right") && Input.GetKey("left") && b
- [x] Input.GetKey("left") && b
- [x] b
- [ ] a
- [ ] Input.GetKey("right") && a

条件式には「true」か「false」の形になるものだけ入れることが出来ます。 また「&&」を使って条件を増やすことが出来ます。

【Step2】09_壁だけに触れている時はジャンプ出来ないようにしよう 小テスト

1. **Playerの中にスプライト（画像）を作成し、Playerの足元に表示されるように設定しました。 その時の説明として、正しいものを全て選びなさい。**

- [x] Playerを移動させると、Playerの中のスプライトも一緒に移動する
- [ ] Playerが「子」で、Playerの中のスプライトは「親」である
- [x] Playerを動かした時、スプライトのTransformの位置の値は変わらない

この場合、Player が「親」で、Player の中のスプライトは「子」となります。 「子」は「親」と一緒に移動するので、Player の中のスプライトも Player と一緒に移動します。 「子」の Transform の位置は、「親」からの位置になります。 「親」を動かしても「親」から見た「子」の位置は変わらないので、「子」の Transform の位置の値は変わりません

2. **Triggerに関する以下の説明で、正しいものを全て選びなさい。**

※○○には「Enter2D」や「Exit2D」などが入ります

- [ ] TriggerとColliderの当たり判定を取得したい場合は「OnCollision○○」を使用する
- [x] Triggerを使うには、Colliderのコンポーネントをアタッチして、TriggerをONにする
- [x] TriggerをONにするとCollider（壁や地面）にぶつからなくなる
- [x] TriggerとTriggerの当たり判定を取得したい場合は「OnTrigger○○」を使用する

「OnCollision○○」を使うと、「Collider」同士の当たり判定を取得できます。 Trigger と Collider の当たり判定は、動画でやったように「OnTrigger○○」で取得できます。 ただしこういった当たり判定には、少なくとも片方に「静的」でない Rigidbody が付いている必要があります。 当たり判定がうまくいかない場合は、Rigidbody が付いているか確認しましょう。