

BallMaze (ボール迷路)

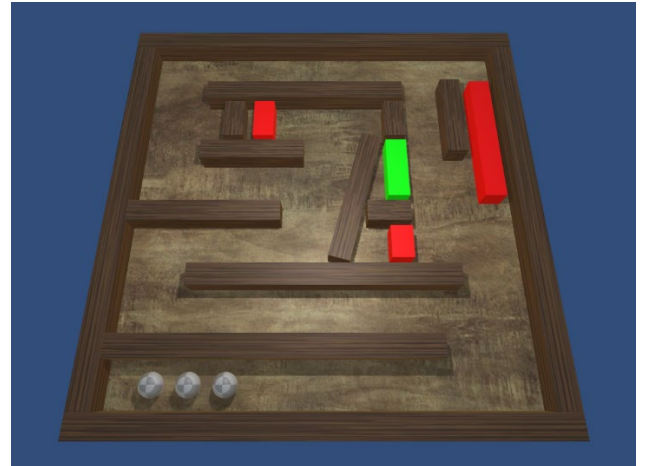
ボールを転がしてゴールまで運ぶゲームです。キーボードの方向キーで操作します。

ボールは3つあり、3つとも同時にゴール(緑)に入れた時点でゲームクリアとなります。何秒でクリアできるか？を競う内容です。

また、トラップ(赤)に接触すると、スタート地点まで戻されてしまいます。ピンボールなどでも落とし穴に入ったらスタートへ戻されるギミックがあり、リスポーン(再び産む)と呼ばれるゲームの基本的な概念です。

ボールが転がったり、トラップへの接触を判定したり、Unity の物理演算機能が効いてくるコンテンツとなります。

(この迷路デザインは作例です。)



この制作指示書では、今まで履修した僅かな内容だけでもゲームが構成できることを紹介すべく準備したものであり、最低限度の迷路ゲームの基本構造のみで終了となります。

迷路の複雑さは無い状態で終了するので、完成後は、各人でオリジナルの迷路のデザインに取り組みます。

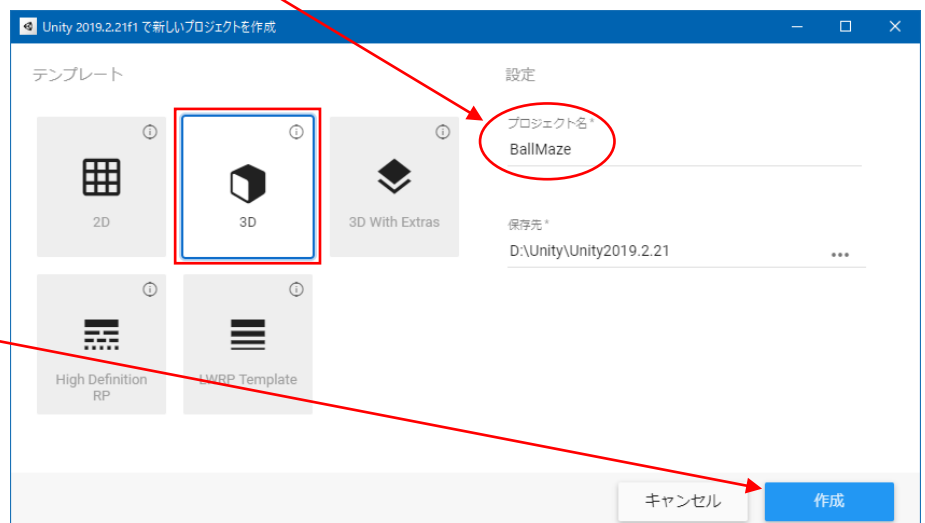
ただ、幾つかの制限が出ていますので、制限指示に合致した中で、最大の創意工夫を発揮して、自由に迷路をデザインして下さい。

プロジェクトの準備

【STEP1】プロジェクトの新規作成

- Unity を起動し、新しいプロジェクト **BallMaze(ボールメイズ)** を 3D モードで準備します。

ボタン**作成**を押下します。



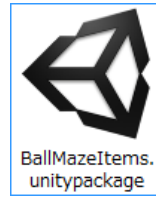
- Game 画面のサイズを **Standalone(1024x768)スタンドアロン** に設定します。



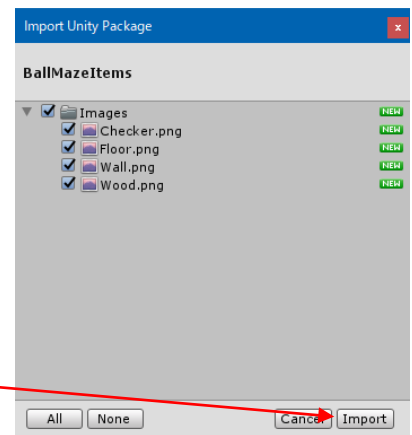
【STEP2】 素材の読み込み

- 配布された素材を読み込みます。

メニューAssets(アセット)から Import Package(インポートパッケージ) > **Custom Package(カスタムパッケージ)**と進み、今回のフォルダ内にある **BallMazeItems.unitypackage** を指定します。

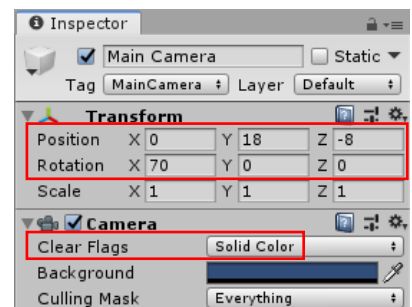


内容物が表示されたら、ボタン **Import(インポート)**を押下します。

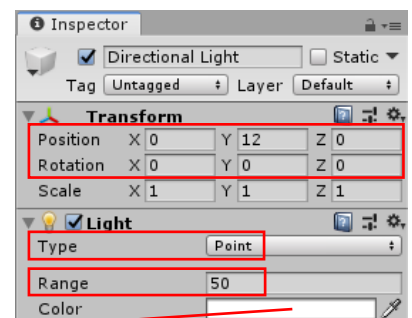


【STEP3】 カメラと光源の設定

- ヒエラルキー欄の **Main Camera(メインカメラ)**を選択し、インスペクタでパラメータを設定します。



- ヒエラルキー欄の **Directional Light(ディレクショナルライト: 指向性照明)**を選択し、インスペクタでパラメータを設定します。

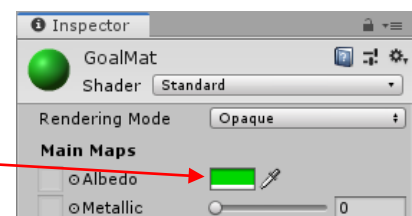


少し黄色っぽいので白にします。

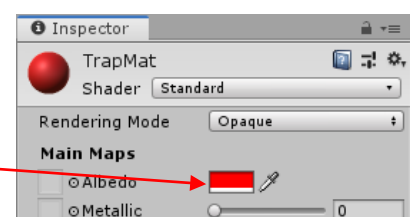


【STEP4】 マテリアル(質感)の準備

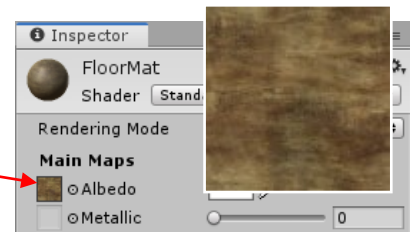
- プロジェクト欄の Create(クリエイト)から **Material(マテリアル)**を作成します。名称は **GoalMat(ゴールマット)**とします。
- インスペクタで緑色を設定します。



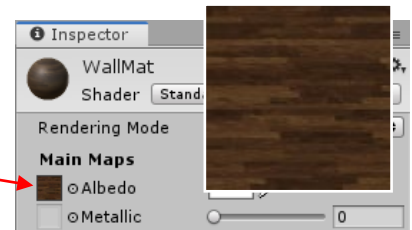
- プロジェクト欄の Create(クリエイト)から **Material(マテリアル)**を作成します。名称は **TrapMat(トラップマット)**とします。
- インスペクタで赤色を設定します。



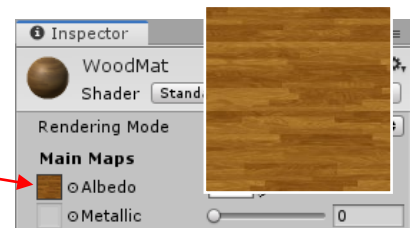
- プロジェクト欄の Create(クリエイト)から **Material(マテリアル)**を作成します。名称は **FloorMat(フロアマット)**とします。
- インスペクタで Albedo(アルベド)の横の○マークを押下し、**画像 Floor(フロア)**をダブルクリックで指定します。



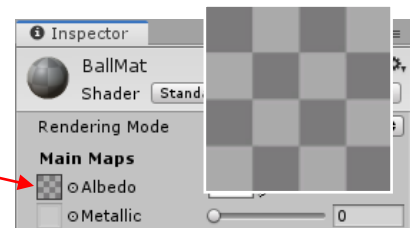
- 同様に、プロジェクト欄の Create(クリエイト)から **Material(マテリアル)**を作成します。名称は **WallMat(ウォールマット)**とします。
- Albedo(アルベド)画像に **Wall(ウォール)**を指定します。



- 同様に、プロジェクト欄の Create(クリエイト)から **Material(マテリアル)**を作成します。名称は **WoodMat(ウッドマット)**とします。
- Albedo(アルベド)画像に **Wood(ウッド)**を指定します。



- 同様に、プロジェクト欄の Create(クリエイト)から **Material(マテリアル)**を作成します。名称は **BallMat(ボールマット)**と命名します。
- Albedo(アルベド)画像に **Checker(チェッカー)**を指定します。

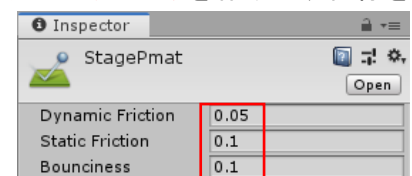


【STEP5】 物理マテリアルの準備

- プロジェクト欄の Create(クリエイト)から **Physic Material(フィジックマテリアル)**を作成し、名称を **StagePmat**とします。

次のパラメータを設定します。

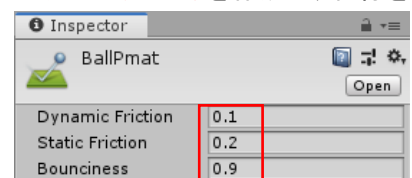
- 動摩擦係数 : 0.05
- 静止摩擦係数 : 0.1
- 反射係数 : 0.1



- 同様に、プロジェクト欄の Create(クリエイト)から **Physic Material(フィジックマテリアル)**を作成し、名称を **BallPmat**と命名します。

次のパラメータを設定します。

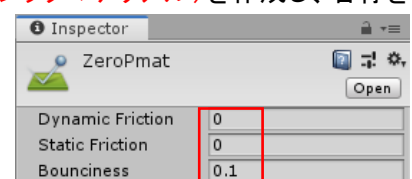
- 動摩擦係数 : 0.1
- 静止摩擦係数 : 0.2
- 反射係数 : 0.9



- 同様に、プロジェクト欄の Create(クリエイト)から **Physic Material(フィジックマテリアル)**を作成し、名称を **ZeroPmat**と命名します。

次のパラメータを設定します。

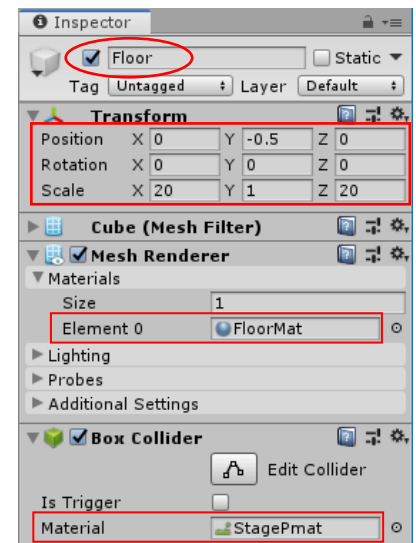
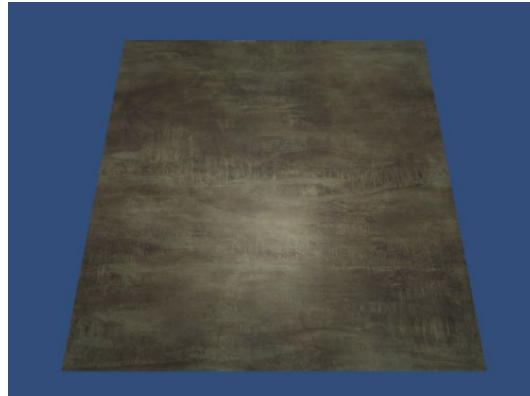
- 動摩擦係数 : 0
- 静止摩擦係数 : 0
- 反射係数 : 0.1



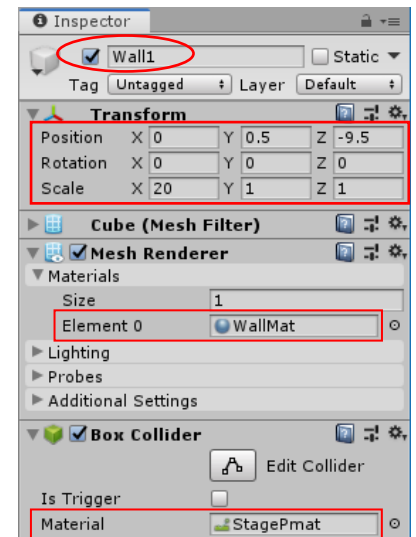
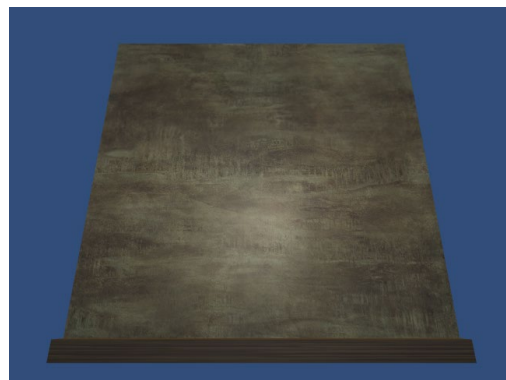
ステージの設営

【STEP6】床と壁の設置

- ヒエラルキー欄の Create(クリエイト) > 3D Object(オブジェクト)> **Cube(キューブ)**と選択して立方体を作成し、名称を **Floor(フロア)**とします。
インスペクタでパラメータを設定します。



- ヒエラルキー欄の Create(クリエイト) > 3D Object(オブジェクト) > **Cube(キューブ)**と選択し、名称を **Wall1(ウォール1)**に変更します。
インスペクタでパラメータを設定します。

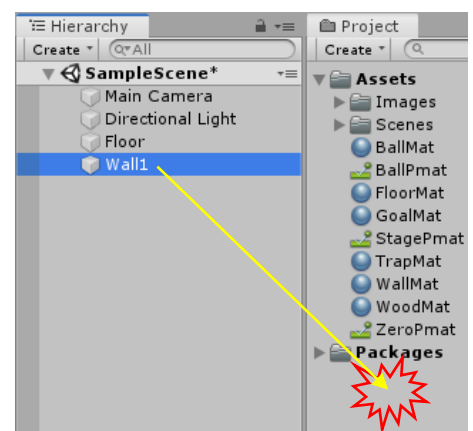


- ヒエラルキー欄の **Wall1** をプロジェクト欄にドラッグ & ドロップします。

「プレハブ」として登録した状況となります。



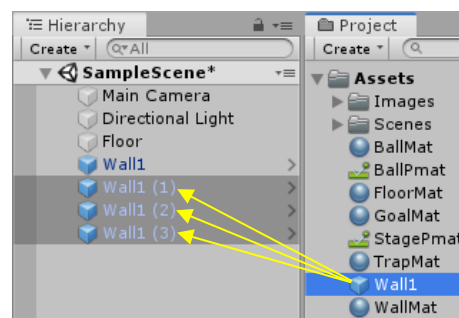
「プレハブ」は幾つ登場させても、メモリ上は1個で済むので、高効率となります。



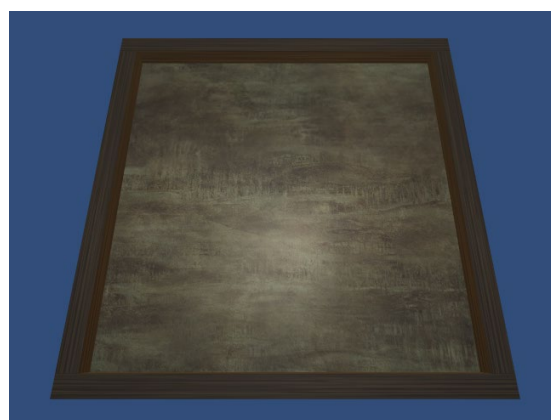
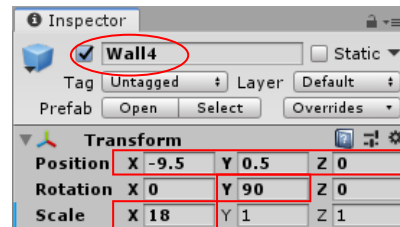
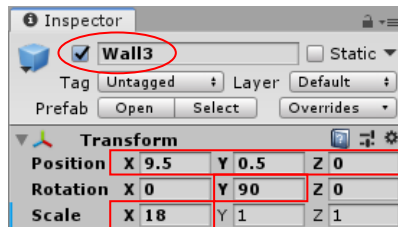
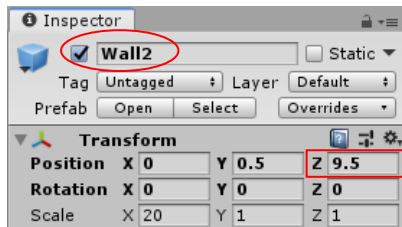
- では、本当に複数個を登場させます。

プレハブ Wall1 を、ヒエラルキー欄に3回ドラッグ & ドロップします。

名称を Wall2~4 に変更します。



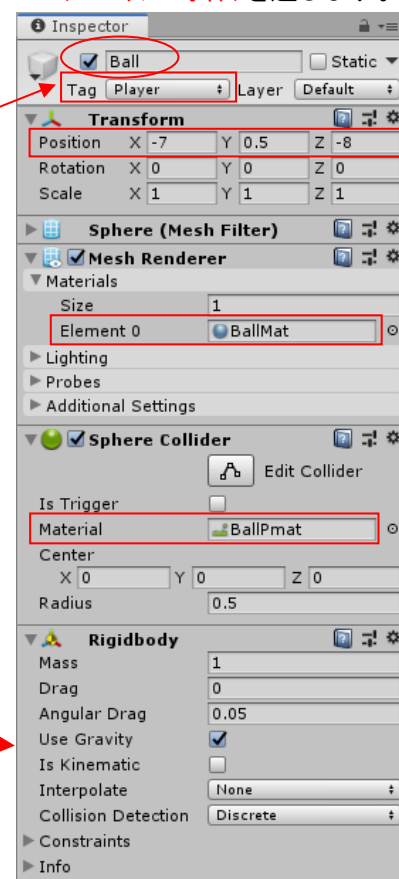
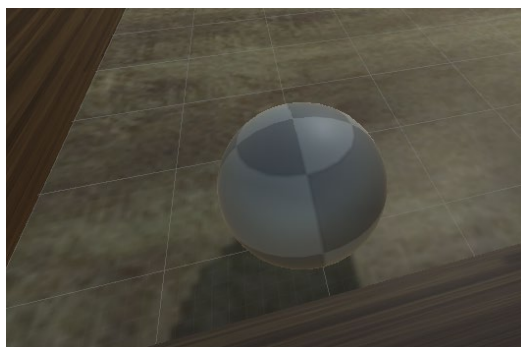
- インスペクタでパラメータを整えます。床の四辺に壁が完成します。



【STEP7】 ボールの設置

- ヒエラルキー欄の Create(クリエイト) > 3D Object(オブジェクト) > Sphere(スフィア:球体)を選びます。
名称を Ball(ボール)とし、タグを Player に設定します。
インスペクタでパラメータを整えます。

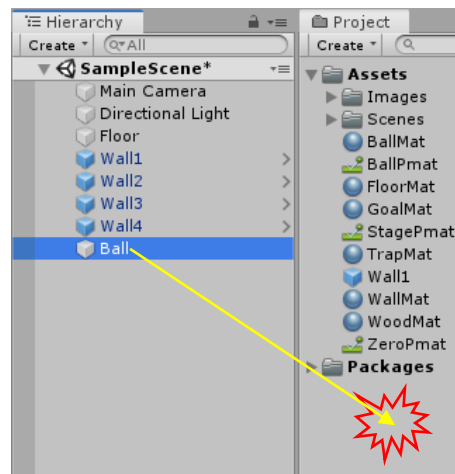
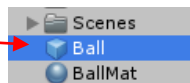
タグの設定し忘れが非常に多いです。



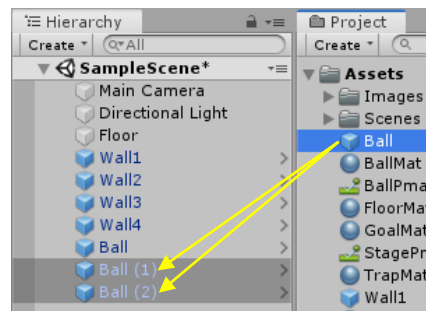
- インスペクタの Add Component から Physics > Rigidbody を選択します。パラメータはそのままです。

- ヒエラルキー欄から Ball をプロジェクト欄へドラッグ & ドロップします。

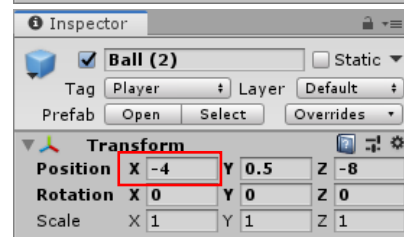
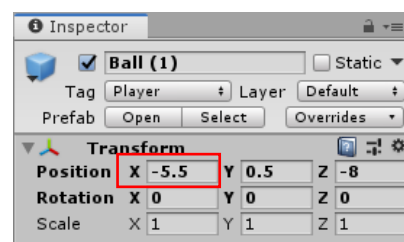
ボールはプレハブ化されました。



- プレハブ化した Ball をヒエラルキー欄へ2回ドラッグ & ドロップして、計3個にします。名称はそのまま構わないでしょう。

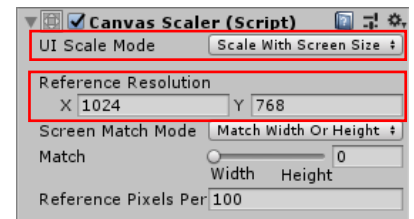


- 追加したボール2個の座標位置を設定します。

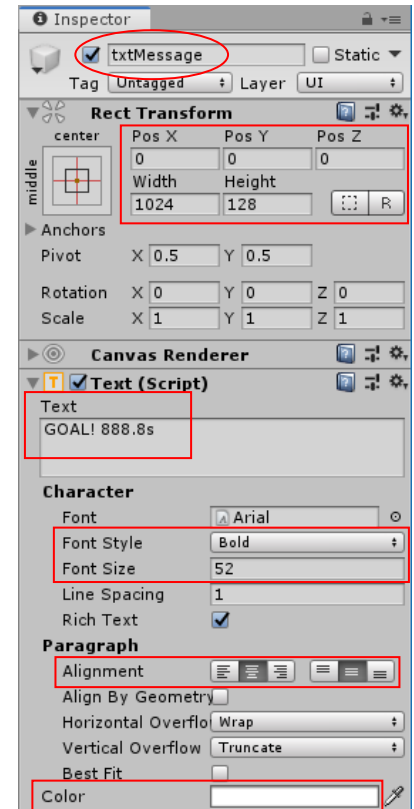


【STEP8】 ユーザーインターフェースの作成

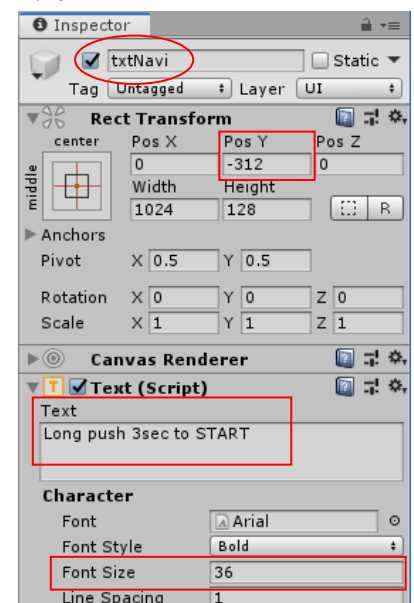
- ヒエラルキー欄の Create(クリエイト)から UI > **Text(テキスト)**を選択します。
- 同時にヒエラルキー欄に **Canvas(キャンバス)**が出来ています。
これを選択し、インスペクタでパラメータを設定します。
Canvas Scaler のコンポーネントです。



- ヒエラルキー欄の Text を選択し、**txtMessage** と命名します。
インスペクタでパラメータを整えます。



- ヒエラルキー欄の **txtMessage** を複製(Ctrl + D)し、名称を **txtNavi** とします。
インスペクタでパラメータを整えます。



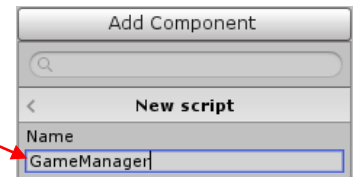
【STEP9】 ゲームの状態管理

ゲームには**状態の管理**が必要です。今回の非常に小規模なゲームでも、3つの状態を扱います。



- タイトル画面では、文字を点滅させてゲーム開始を促します。まだボールが動かせない状況です。
- プレイ中は、わざと経過時間を表示しないようにします。最後のお楽しみです。
- ゴールの条件が成立したらクリア画面に進み、所要時間を表示します。ボール操作は再び禁じます。
- クリア画面で長押し3秒を行うと、タイトル画面に戻るものとします。

- ヒエラルキー欄の **Main Camera** を選択し、インスペクタの Add Component から **New script** を選択します。名称を **GameManager** とします。



- スクリプト **GameManager** を次のように編集します。

列挙型変数:enum

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; //uGUIを利用するのに必要
using UnityEngine.SceneManagement; //リロードに必要な

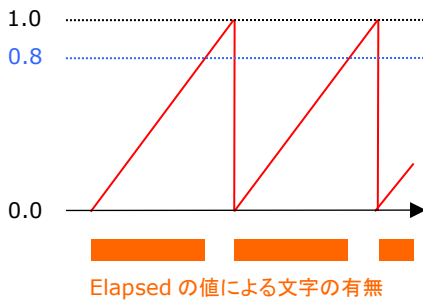
public class GameManager : MonoBehaviour {

    public Text txtMessage; //メッセージ
    public Text txtNavi; //ナビ
    public enum STS {
        TITLE, //タイトル画面
        PLAY, //プレイ画面
        CLEAR //クリア画面
    }
    STS GameStatus; //ゲームの状態
    float Elapsed; //経過時間

    void Start() {
        GameStatus = STS.TITLE; //タイトル画面
        txtMessage.text = "Ball Maze";
        Elapsed = 0.0f; //ゼロクリア
    }

    //続きます
```

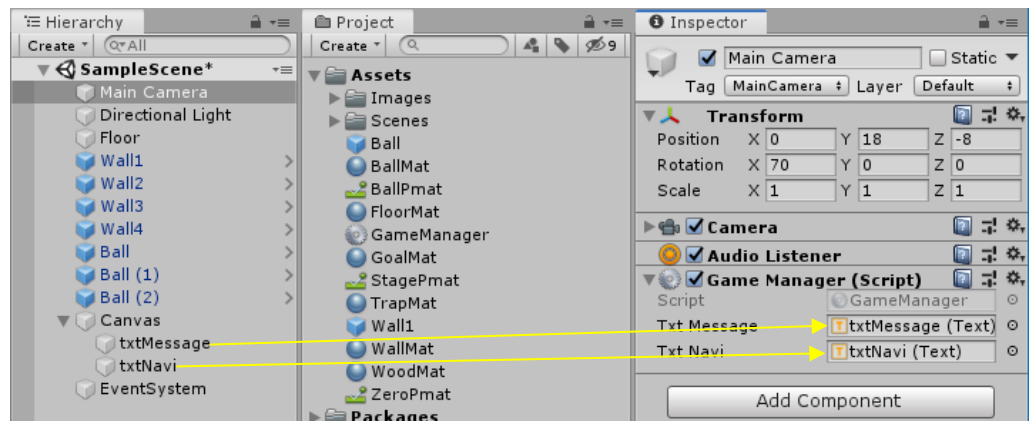

2分岐ではなく、多分岐の構造を実現する制御文:switch~case



//続きます

```
void Update() {
    Elapsed += Time.deltaTime; //経過時間を加算
    switch (GameStatus) {
        case STS.TITLE: //タイトル画面
            if (Elapsed < 1.0f) {
                if (Elapsed < 0.8f) {
                    txtNavi.text = "Long push 3sec to START";
                } else {
                    txtNavi.text = "";
                }
            } else {
                Elapsed = 0.0f;
            }
            break;
        case STS.PLAY: //プレイ画面
            break;
        case STS.CLEAR: //クリア画面
            break;
        default: //状態エラー
            break;
    }
}
```

- ヒエラルキー欄の **Main Camera** を選択し、インスペクタに登場した項目をドラッグ & ドロップします。



- プレイボタンを押下します。



テキストが点滅することを確認します。



【STEP10】記述の効率化

前述のテキスト点滅の運営に於いて、結果は同じなのですが、もう少し効率的にスマートに記述することが出来ます。書き換え問題みたいですが、if文を中心に実際に書き換えます。

- スクリプト **GameManager** を次のように編集します。

if 文が一つ減っています。

```
//～前略～
void Update() {
    Elapsed += Time.deltaTime; //経過時間を加算
    switch (GameStatus) {
        case STS.TITLE: //タイトル画面
            Elapsed %= 1.0f;
            if (Elapsed < 0.8f) {
                txtNavi.text = "Long push 3sec to START";
            } else {
                txtNavi.text = "";
            }
            break;
        case STS.PLAY: //プレイ画面
            break;
    }
}
//～後略～
```

- プレイボタンを押下します。



短縮化して記述しても変化がないことを確認します。



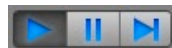
プログラム記述のテクニックで、3項演算子を用いると、更に効率的に記述できる部位があります。

- スクリプト **GameManager** を次のように編集します。

if 文はゼロです。

```
//～前略～
void Update() {
    Elapsed += Time.deltaTime; //経過時間を加算
    switch (GameStatus) {
        case STS.TITLE: //タイトル画面
            Elapsed %= 1.0f;
            txtNavi.text = ( Elapsed < 0.8f ) ? "Long push 3sec to START" : "";
            break;
        case STS.PLAY: //プレイ画面
            break;
    }
}
//～後略～
```

- プレイボタンを押下します。



一応、変化がないことを確認します。



【STEP11】 開始を検出してプレイ操作へ

- スクリプト **GameManager** を次のように編集します。

```
//～前略～
float LongPush; //長押し時間

void Start() {
    GameStatus = STS.TITLE; //タイトル画面
    txtMessage.text = "Ball Maze";
    Elapsed = 0.0f; //ゼロクリア
    LongPush = 0.0f; //ゼロクリア
}

void Update() {
    Elapsed += Time.deltaTime; //経過時間を加算
    switch (GameStatus) {
        case STS.TITLE: //タイトル画面
            Elapsed %= 1.0f;
            txtNavi.text = ( Elapsed < 0.8f ) ? "Long push 3sec to START" : "";
            if (Input.GetMouseButton( 0 )) {
                LongPush += Time.deltaTime;
                //画面を3秒長押しでゲーム開始
                if (LongPush > 3.0f) {
                    GameStatus = STS.PLAY;
                    txtMessage.text = "";
                    txtNavi.text = "";
                    Elapsed = 0.0f; //プレイ時間初期化
                    LongPush = 0.0f; //ゼロクリア
                }
            } else {
                LongPush = 0.0f; //指を離れた
            }
            break;
        case STS.PLAY: //プレイ画面
            Vector3 Dir = new Vector3( 0, -1, 0 );
            Dir.x = Input.GetAxis( "Horizontal" );
            Dir.z = Input.GetAxis( "Vertical" );
            Physics.gravity = 9.81f * Dir.normalized;
            break;
        case STS.CLEAR: //クリア画面
            break;
        default: //状態エラー
            break;
    }
}
}
```

ユーザーの長押し3秒を監視する。

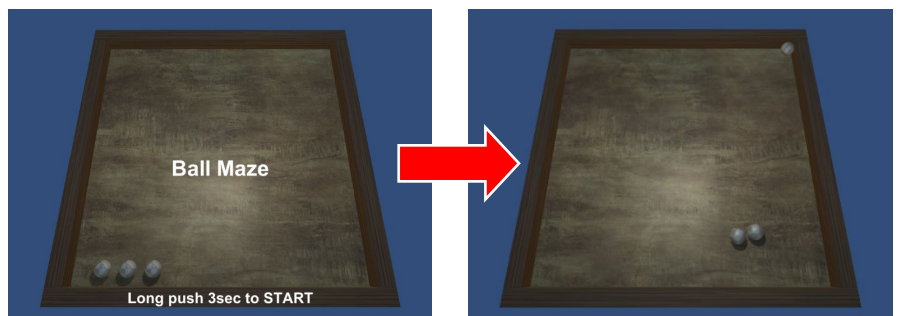
上下左右キーの操作で重力方向を変化させる。

- プレイボタンを押下します。



開始後は点滅だけです。キーボードの方向キーを押下しても、ボールは転がらないことを確認し、ゲーム画面内を**マウスで長押し3秒**します。

すると、キーボードの方向キーで重力の方向が変わり、それによってボールが転がるようになります。

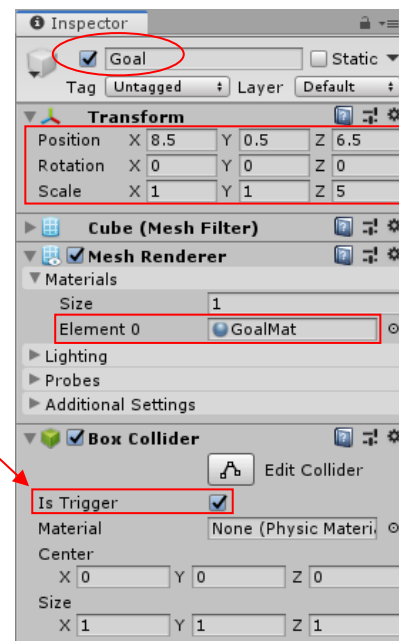
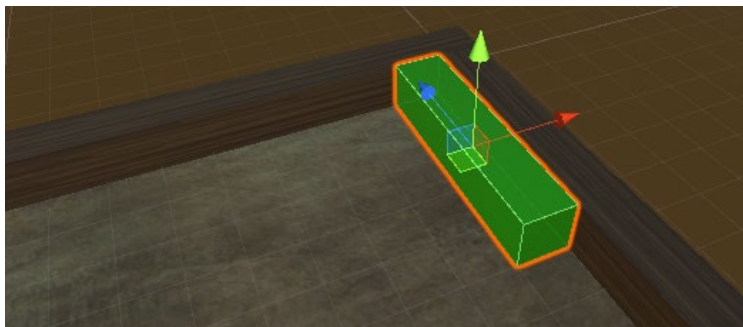


【STEP12】ゴールを設置する

- ヒエラルキー欄の Create から 3D Object(オブジェクト) > **Cube(キューブ)**を選び、名称を **Goal(ゴール)**とします。
インスペクタでパラメータを設定します。

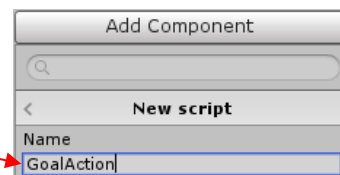
Box Collider(ボックスコライダー)コンポーネントの **Is Trigger(イズトリガー)**をチェックします。

反射系ではなく、侵入系の物理反応となります。



- この Goal を選択している状態で、インスペクタの Add Component から **New script** を選択します。名称を **GoalAction** とします。

- スクリプト **GoalAction** を次のように編集します。



int 型の前に **static public** を付加すると、外部つまり GameManager からでも参照や更新が可能になります。

自身にボールが侵入すれば1加算、出て行けば1減算しています。

タグが Player かを確認しないと、床や壁でも接触が発生してカウントしてしまいます。

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

public class GoalAction : MonoBehaviour {

    static public int Cnt; //自身(Goal)に侵入しているボール数

    void Start() {

    }

    private void OnTriggerEnter(Collider other) {
        if (other.gameObject.tag == "Player") {
            Cnt++;
        }
    }

    private void OnTriggerExit(Collider other) {
        if (other.gameObject.tag == "Player") {
            Cnt--;
        }
    }

    void Update() {

    }

}
```

- スクリプト **GameManager** を次のように編集します。

```
//～前略～
int AllBalls; //ボールの総数

void Start() {
    //ボールの総数を求める
    AllBalls = GameObject.FindGameObjectsWithTag( "Player" ).Length;
    GoalAction.Cnt = 0; //Goalへの侵入ボール数をゼロクリア
    GameState = STS.TITLE; //タイトル画面
    txtMessage.text = "Ball Maze";
    Elapsed = 0.0f; //ゼロクリア
    LongPush = 0.0f; //ゼロクリア
}

void Update() {
    Elapsed += Time.deltaTime; //経過時間を加算
    switch (GameState) {
        case STS.TITLE: //タイトル画面
            Elapsed %= 1.0f;
            txtNavi.text = ( Elapsed < 0.8f ) ? "Long push 3sec to START" : "";
            if (Input.GetMouseButton( 0 )) {
                LongPush += Time.deltaTime;
                //画面を3秒長押しでゲーム開始
                if (LongPush > 3.0f) {
                    GameState = STS.PLAY;
                    txtMessage.text = "";
                    txtNavi.text = "";
                    Elapsed = 0.0f; //プレイ時間初期化
                    LongPush = 0.0f; //ゼロクリア
                }
            } else {
                LongPush = 0.0f; //指を離れた
            }
            break;
        case STS.PLAY: //プレイ画面
            Vector3 Dir = new Vector3( 0, -1, 0 );
            Dir.x = Input.GetAxis( "Horizontal" );
            Dir.z = Input.GetAxis( "Vertical" );
            Physics.gravity = 9.81f * Dir.normalized;
            if (GoalAction.Cnt >= AllBalls) {
                Debug.Log("Cleared!");
                GameState = STS.CLEAR;
            }
            break;
    }
}

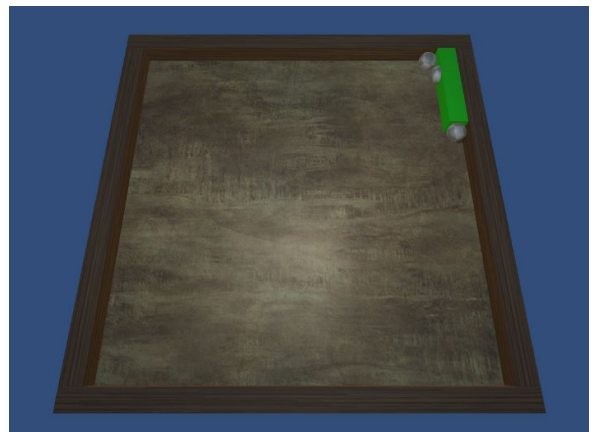
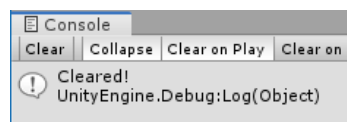
//～後略～
```

プレイ中に、ゴール侵入ボール数が総数に至ったらメッセージを出す。

- プレイボタンを押下します。



ボールを操作できるようになったら、ゴールへ侵入させます。コンソールに文字 **Cleared!** が表示されます。



【STEP13】 クリア画面の運営

ゲームクリア時は所要時間の表示と、タイトル画面に戻る為の待ち受け(画面長押し3秒)を行います。

- スクリプト **GameManager** を次のように編集します。

リロードしても重力は
変位したままなので、
リセットする。

現在のシーンをリロー
ド再生する。

```
//～前略～
case STS.PLAY: //プレイ画面
    Vector3 Dir = new Vector3( 0, -1, 0 );
    Dir.x = Input.GetAxis( "Horizontal" );
    Dir.z = Input.GetAxis( "Vertical" );
    Physics.gravity = 9.81f * Dir.normalized;
    if (GoalAction.Cnt >= AllBalls) {
        Debug.Log("Cleared!");
        GameStatus = STS.CLEAR;
        txtMessage.text = "GOAL! " + Elapsed.ToString( "f2" ) + "s";
        Physics.gravity = new Vector3( 0, -9.81f, 0 ); //重力リセット
        LongPush = 0.0f; //長押しクリア
    }
    break;
case STS.CLEAR: //クリア画面
    Elapsed %= 1.0f;
    txtNavi.text = ( Elapsed < 0.8f ) ? "Long push 3sec to TITLE" : "";
    if (Input.GetMouseButton( 0 )) {
        LongPush += Time.deltaTime;
        //画面を3秒長押しで現在シーンをリロード
        if (LongPush > 3.0f) {
            SceneManager.LoadScene( gameObject.scene.name );
        }
    } else {
        LongPush = 0.0f; //指を離れた
    }
    break;
default: //状態エラー
    break;
}
```

- プレイボタンを押下します。



ボール3個を操作して、同時にゴールオブジェクト内に入れます。

ゲームプレイに所要した時間が表示されます。

ゴールしてからは、キーボードの操作を受け付けなくなります。

画面の長押しを3秒間すると、シーンがリロードされることを確認します。

これで何度も遊べる最低限度の状態が構成できました。これをシーケンスが回った状態になった、と言います。

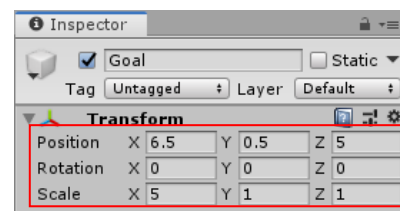


ギミック(仕掛け)の運営

ゲームの遊興性を左右するギミック(仕掛け)を作成します。簡単に言うと、簡単にゴールはさせないぞ！というルールを設けることになります。ここでは、①ボール動作と同調してスライドしてしまう壁と、②当たってしまうと指定位置にワープ(リスポーン)してしまう壁を設けます。

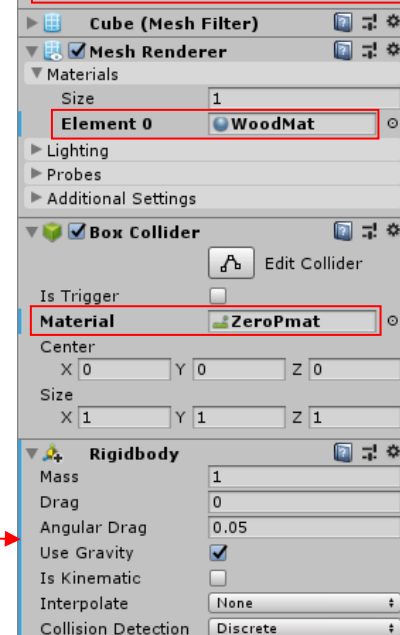
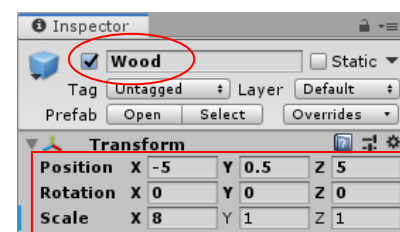
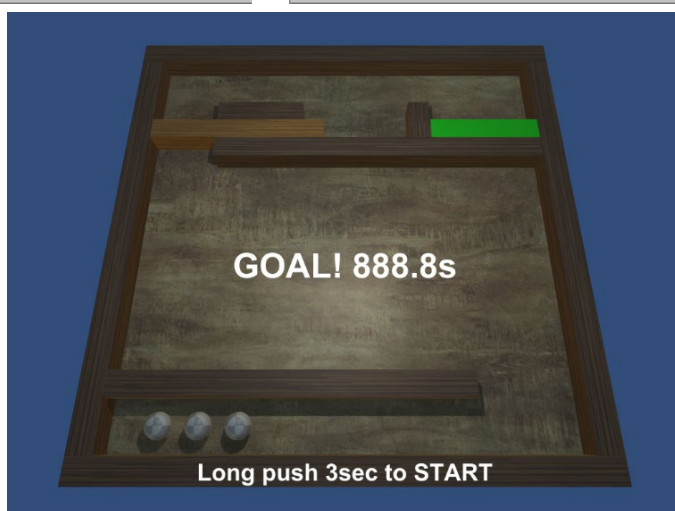
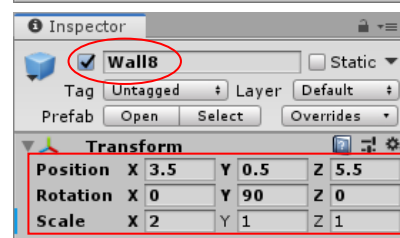
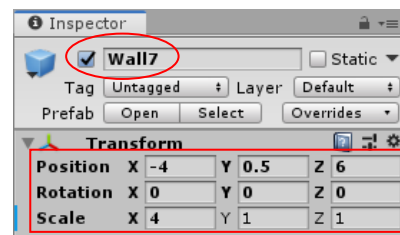
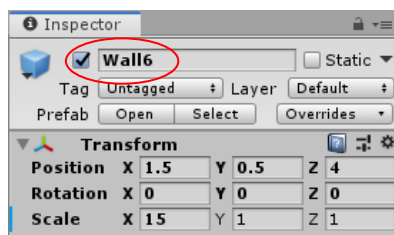
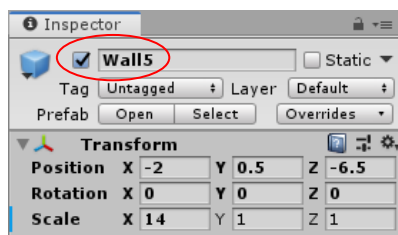
【STEP14】 スライドする壁

- ヒエラルキー欄の **Goal** を選択し、インスペクタでパラメータを設定します。



- ヒエラルキー欄の **Wall1** を選択し、複製操作(Ctrl + D)を5回行います。名称を **Wall5**、**Wall6**、**Wall7**、**Wall8**、**Wood** とします。

インスペクタでパラメータを設定します。



ボール操作に同調し、一緒に床をスライドさせる為、床との摩擦をゼロにします。

- ヒエラルキー欄の Add Component から Physics > **Rigidbody** を選択します。パラメータはそのままです。

- プレイボタンを押下します。



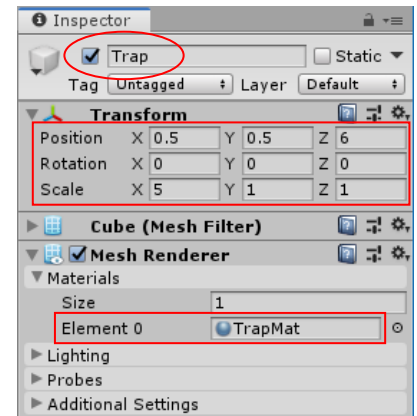
ボールを操作する為だけのキーボード操作であったはずが、この Wood までもがボールと同調して動く為に、コースを塞ぐことになってしまいました。

少し遊興性が芽生えつつあると言えるギミック(仕掛け)です。

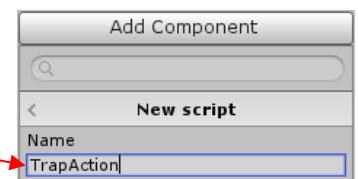


【STEP15】 当たるとボールがワープ(リスポーン)する壁

- ヒエラルキー欄の Create から 3D Object(オブジェクト) > Cube(キューブ)を選び、名称を Trap(トラップ)とします。インスペクタでパラメータを調整します。



- この Trap を選択している状態で、インスペクタの Add Component から New script を選択し、名称を TrapAction とします。



- スクリプト TrapAction を次のように編集します。

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

public class TrapAction : MonoBehaviour {

    public Vector3 RespawnPosition = new Vector3(-8, 0.5f, -8);

    void Start() {

    }

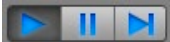
    void OnCollisionEnter(Collision other) {
        if (other.gameObject.tag == "Player") {
            other.gameObject.transform.position = RespawnPosition;
        }
    }

    void Update() {

    }

}
```

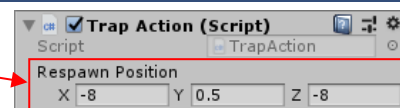
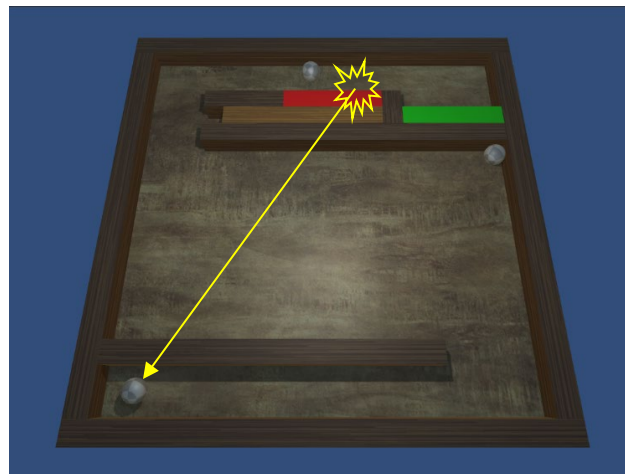
トラップにボールが侵入したら、指定座標を渡すことになります。

- プレイボタンを押下します。 

ボールをトラップに接触させ、そのボールがスタート付近へ移動(リスポーン)されるか？を確認します。

このリスポーン先の座標は Trap(トラップ)オブジェクト本人が所有しています。行き先の座標をインスペクタで変更できるようになっています。

ヒエラルキー欄の Trap(トラップ)を選択し、インスペクタで確認します。

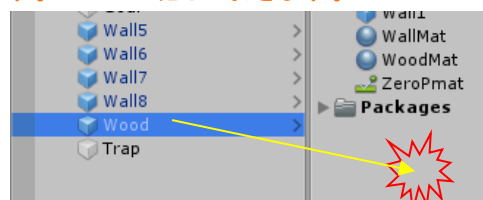


今後の作業で、この Trap(トラップ)を複製した際に、それぞれ別々の行き先(座標)を指定することが可能になっています。

【STEP16】 ギミックの複製対応(プレハブにしておく)

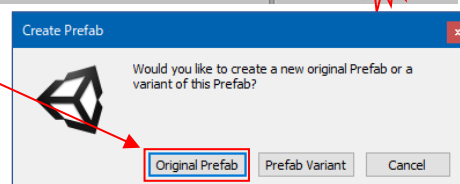
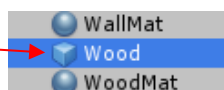
スライドする壁やトラップの壁は、各人で増やしていくことが予想されます。プレハブ化しておきます。

- ヒエラルキー欄の Wood をプロジェクト欄にドラッグ & ドロップします。



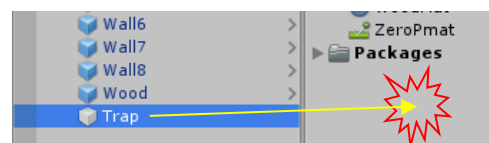
オリジナルプレハブにするか聞かれるので、選んでおきます。

プレハブ化されました。



- ヒエラルキー欄の Trap をプロジェクト欄にドラッグ & ドロップします。

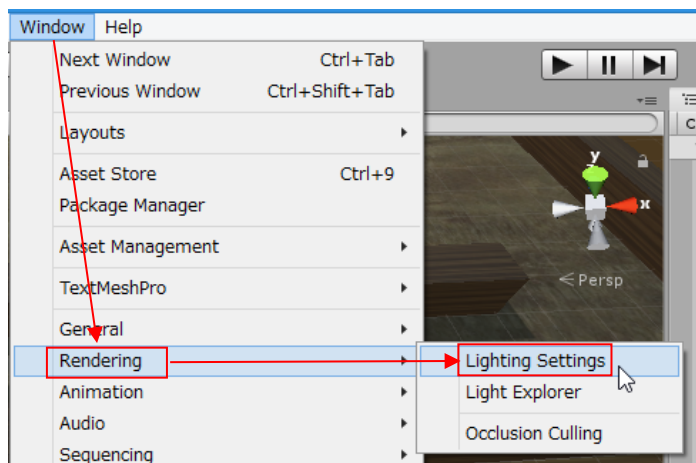
プレハブ化されました。



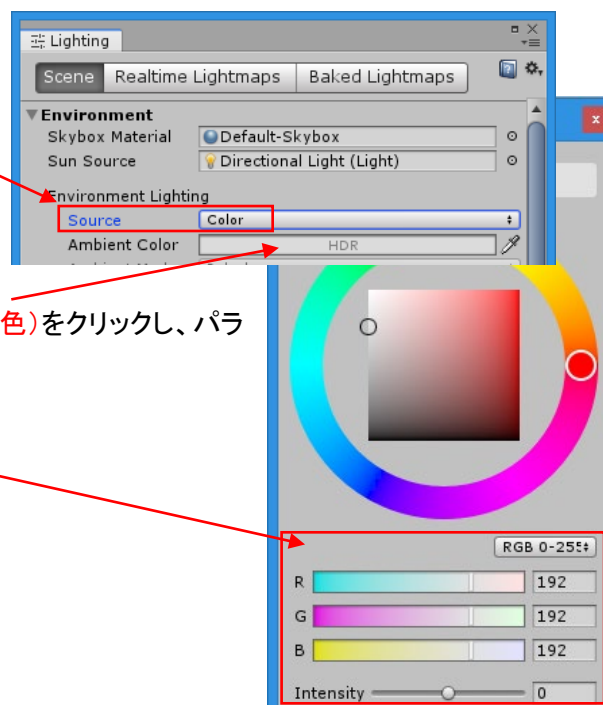
【STEP17】 環境光を設定する

現状では、画面が少し青っぽい感じです。最初の青空が実は残っていて、その青みを環境光として反映して青みを帯びているのです。この世界空間を真っ白にして、物体の色がそのまま出る表現方法に変更します。

- メニューWindow(ウィンドウ)から Rendering(レンダリング) > Lighting Settings(ライティングセッティングス)を選択します。

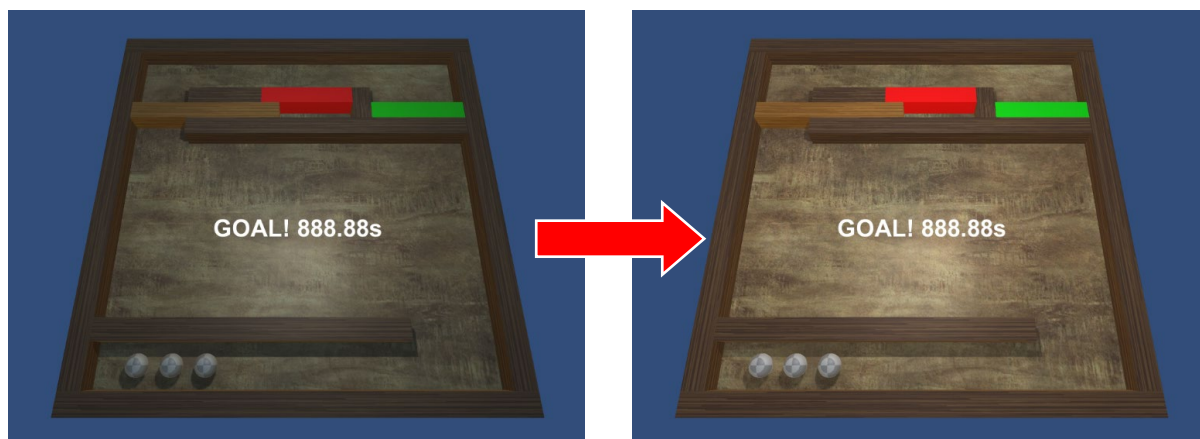


- Environment Lighting(エンバイアメントライティング: 環境光)の Source(ソース:原資)を Color(カラー)にします。



- その次の項目 Ambient Color(アンビエントカラー: 周囲色)をクリックし、パラメータを設定します。

この Lighting(ライティング)パネルは閉じておきます。
明るくなりました。青っぽくもありません。



【STEP18】 迷路をデザインする

現状までの作業で完成したゲーム「BallMaze(ボール迷路)」では、迷路の要素がありません。完全に自由に制作してもいいのですが、成功にはパターンがあります。職業としてのゲーム制作では、他人が面白いと思う遊興性には定番的な法則があり、各会社によっても違いますが、約7割程度までは決まった型通りに作ります。(作曲で例えると、音楽は自由だ！なんて言いますが、バッハが定めた平均律という法則に従わないと、とても聞けたものじゃない！というのと同じです。)

今回の制作の場合、以下のものが相当します。この型に当てはめると、勝手に遊興性が向上します。但し、上手く当てはめるのが難しいのです。各項目を維持しつつ、各人で迷路のデザインを行って下さい。

＜実装項目＞

- ① ゴールまでの道のりができるだけ長く、また興味深いものにする。但し、複雑にし過ぎない。
- ② ゴールへの経路を「難しい近道」と「簡単な遠回り」の2系統にし、ユーザーが選べるようにする。
- ③ それら2系統に共通して、最初は簡単なのに、ゴールに近くなるほど難易度カーブが上昇する。
- ④ プレイヤーに対して秘匿した、隠された仕組みを一切排除する。但し、最初は判らないけれど、プレイするとスグに判る仕組み(ギミック)は歓迎されます。

＜実際の作業としては＞ 改変例を挙げると、以下のようなものが現実的です。

- 迷路となる壁を増やす。(Wall1～8を複製(Ctrl + D)して作る。)位置や大きさ、角度なども変更する。
- トラップの位置や大きさ、数を変更する。また、リスポーン先となる座標位置を興味深いものにする。
- ゴールの位置や大きさを変更する。
- ボールのスタート位置を変更する。3個それぞれを、全く離れた位置にするなど。
- プレイ全体には影響が少ないものの、壁の色や貼ってある画像を変更する。

＜変更不可＞ 以下の既存機能は変更不可とします。

- 床(Floor:フロア)より大きくしない。また、四辺の壁も変更しない。カメラも動かさない。
- 画面タッチしてから時間計測し、ゴール時にタイム表示する仕組み。これは変更しない。
- ボールは3個のままとし、ゴール内に3個が同時突入でクリア判定となる仕組みはそのままとします。
- ゴールは1個とし、ボール3個分が入る大きさを維持します。
- トラップに接触してリスポーンされる座標位置は迷路内部として下さい。

＜作例紹介＞

2系統の通路が用意され、簡単な遠回りと難しい近道になっていることが判ります。

また、最初は簡単で終盤に危険が多く、難易度カーブも良好です。

スライドする壁 Wood が無くても、このように遊興性を達成することが可能です。

こうしたゲームプレイのフィールドをデザインする作業を「レベルデザイン」といい、これだけを行っている専門職もあります。上手なデザインのコツとしては、構築中に自身でも何度もプレイ操作を行って、初めてプレイする人が楽しく思えるか、自身でもクリアできるデザインになっているか、などを確認します。

