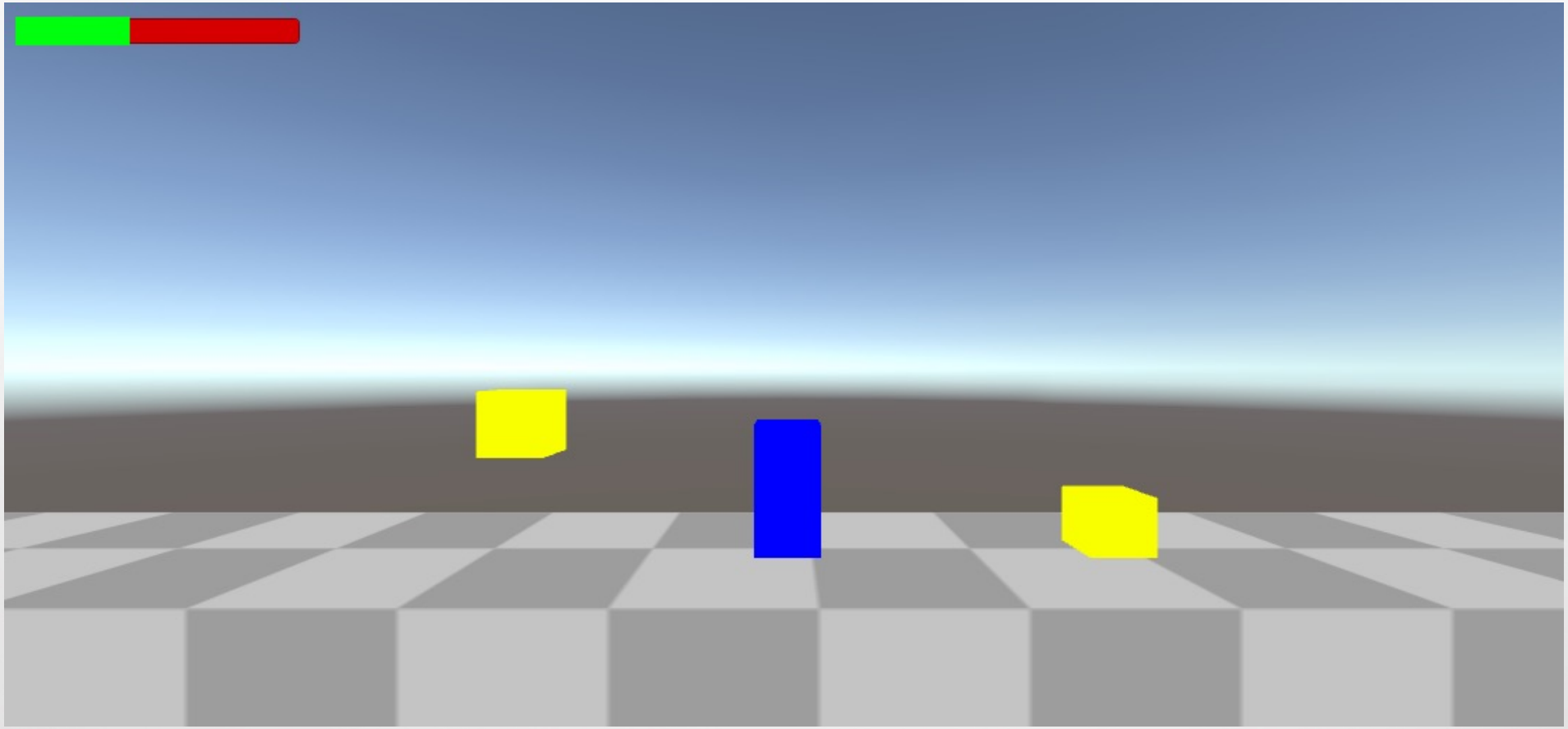
デザインパターン_ステートパターン

目次

０．作成するゲーム



１．ステートパターンとは

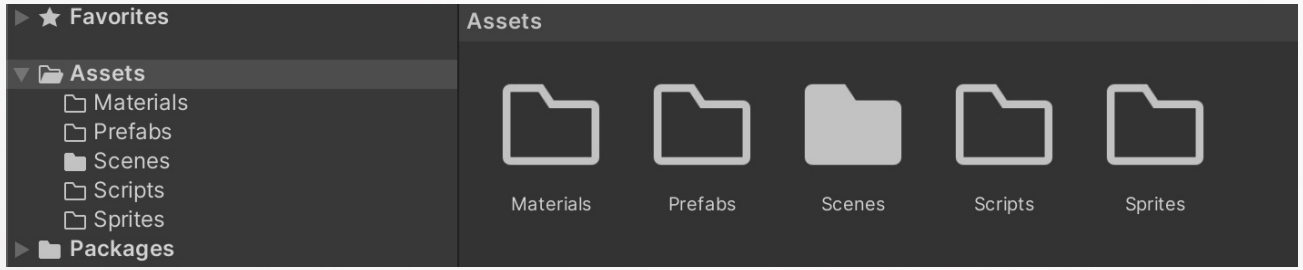
ステートパターン：
ある対象の状態を「enum」ではなく「クラス」で表現し、表現したモノを
切り替えることにより状態の変化を実現すること。
UnityでいうとMecanimみたいなもの！

オブジェクト指向といえば、オブジェクト(モノ)をクラスで表現するが、
ステートパターンでは「状態」をクラスで表現する方法を取っている。
大規模な開発ではもはや必須のデザインパターンの1つとなりつつある。

２．ゲームの作成

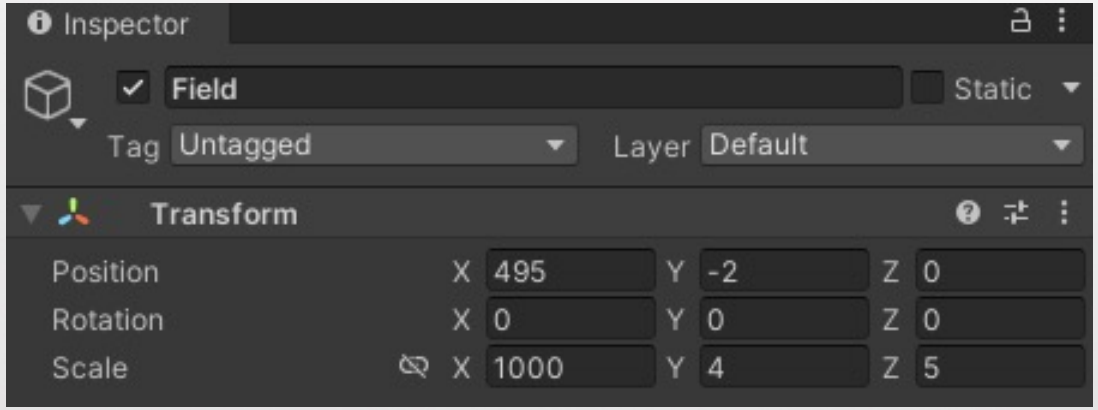
ステートパターンの前に、ゲームの土台を作っていきます。

知識としては今まで実施してきた内容となるので、復習がてらお願いします。



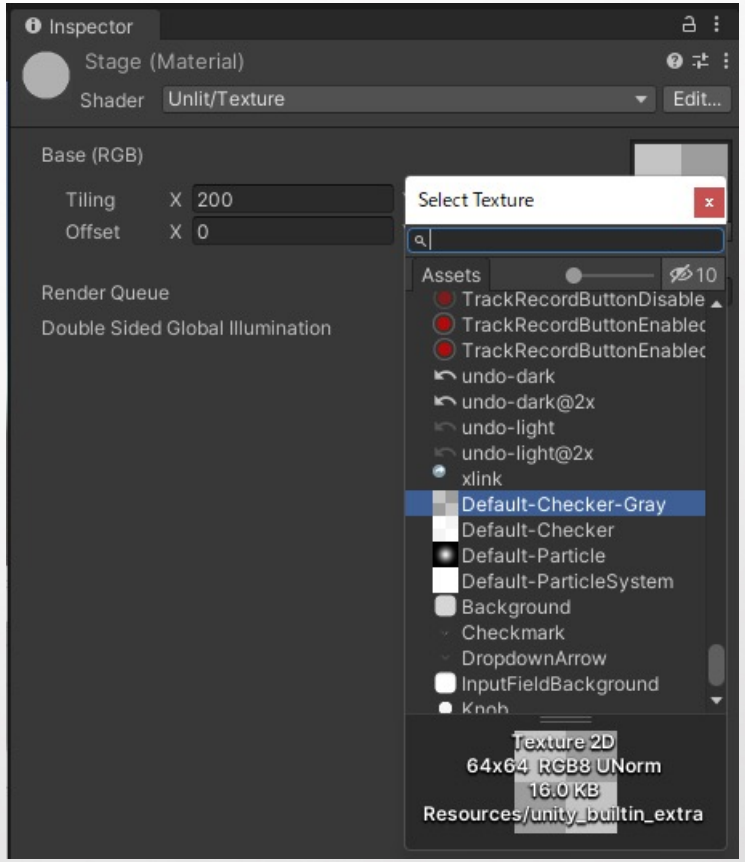
＜オブジェクトの配置＞

１．Cubeを画面に配置、名前は「Field」にし、位置とスケールを調整

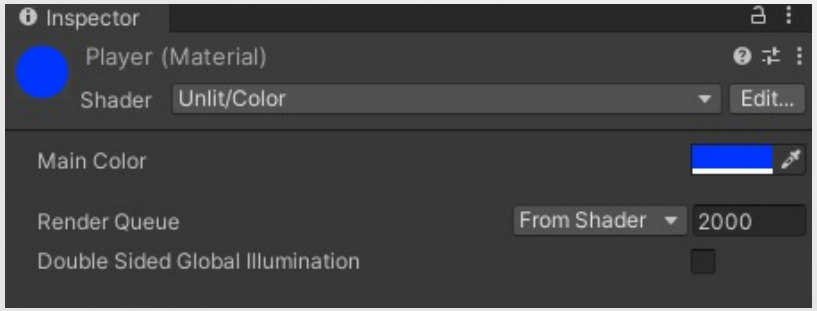


２．ゲームの作成

２．Materialフォルダ内に、新規で「Stage」の名前でマテリアルを作成。
ShaderからUnlit→Textureを設定後、「Default-Checker-Gray」を選択。

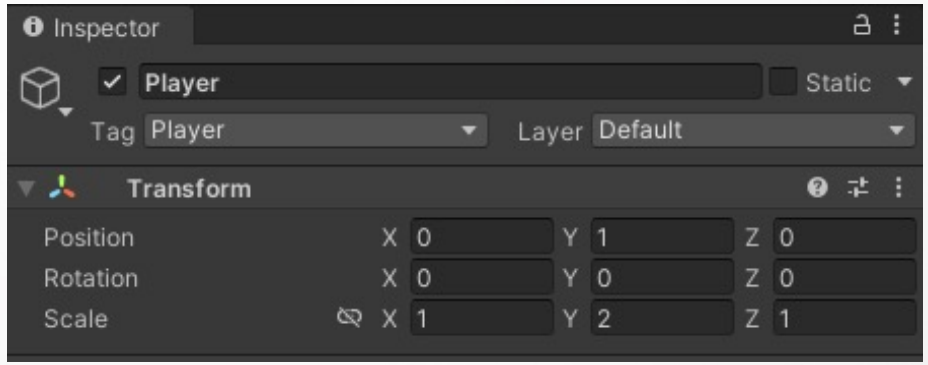


３．併せてもう1つマテリアルを作成、
名前をPlayerとする。
ShaderからUnlit→Colorとし、
青の色を指定する。

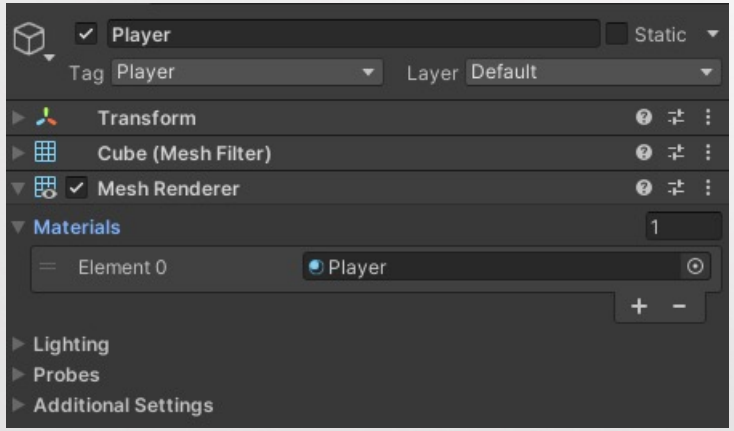


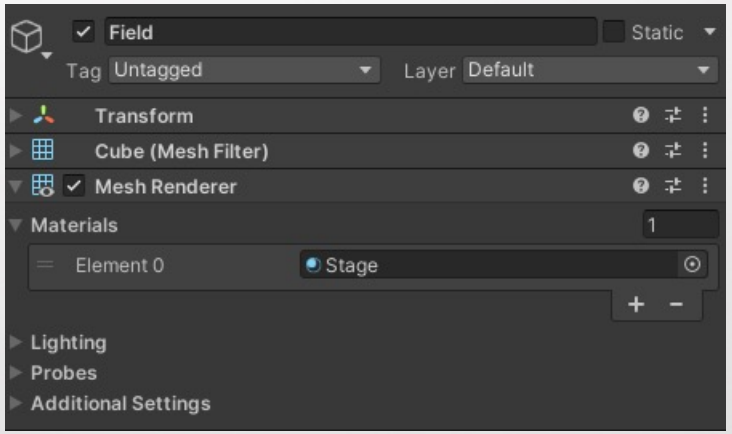
２．ゲームの作成

４．Cubeを新たに作成、「Player」とする。TagとTransformを合わせる。
あとRigidbodyもアタッチ。



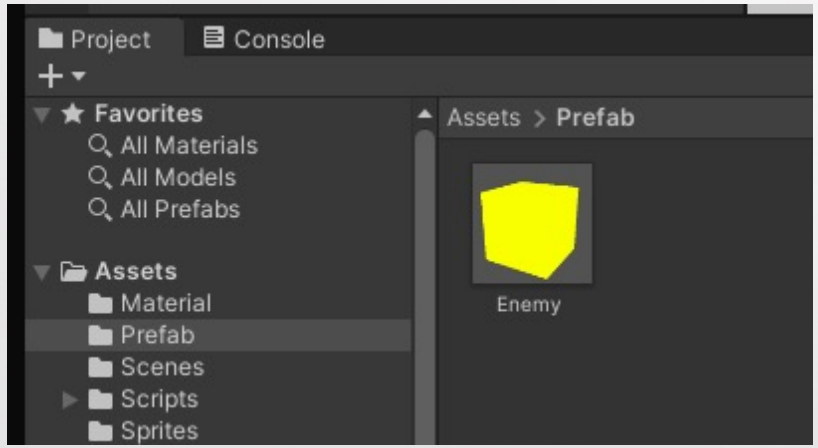
５．PlayerとFieldのそれぞれに、Mesh RendererのMaterialsに
先ほど作ったそれぞれのマテリアルをアタッチする。(Tillingやっても良き)

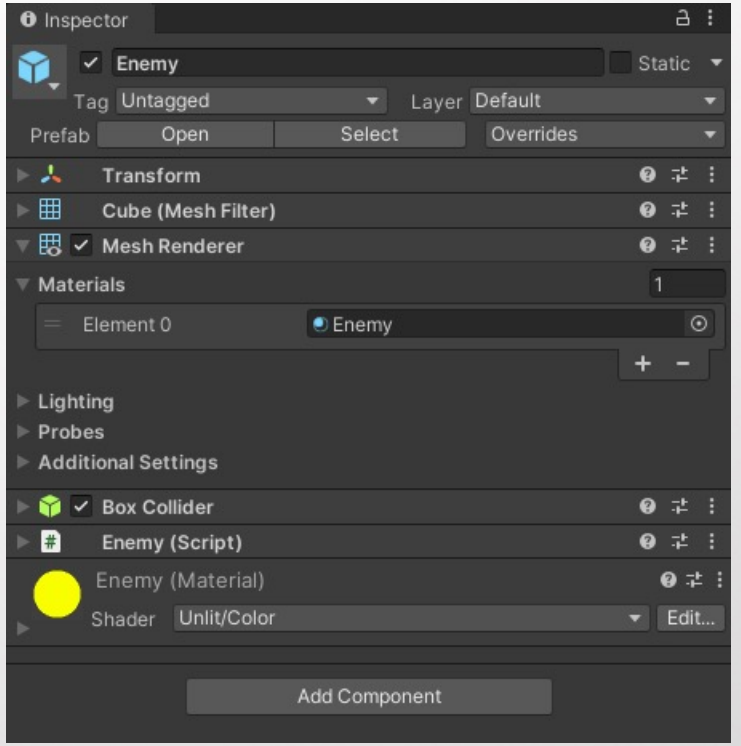




２．ゲームの作成

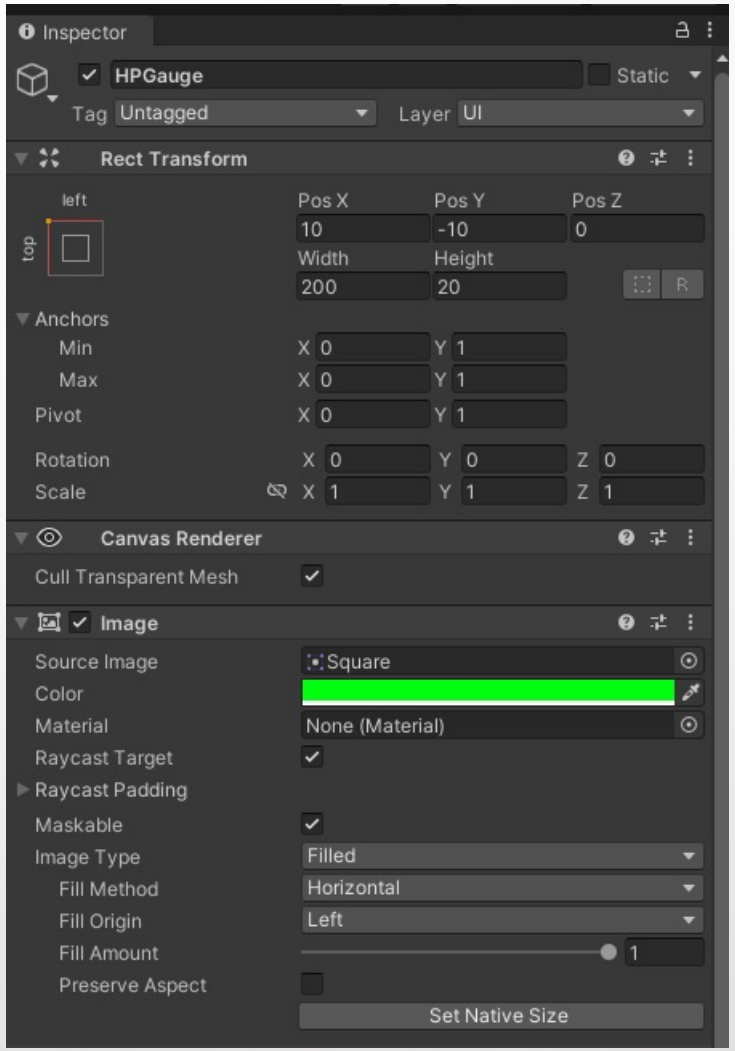
６．　３と同じ要領でMaterialを作成、Enemyとし色は黄色とする。
Cubeを作成し、名前をEnemyと設定し５と同じようにマテリアルを適用する。
マテリアル適応後、Prefabフォルダに格納してヒエラルキー上のEnemyを削除する。
BoxColliderのIsTriggerのチェックONで！





２．ゲームの作成

７．UIからImageを選択、名前は「HPGauge」とする。RectTransformとImageの設定は以下の通り。



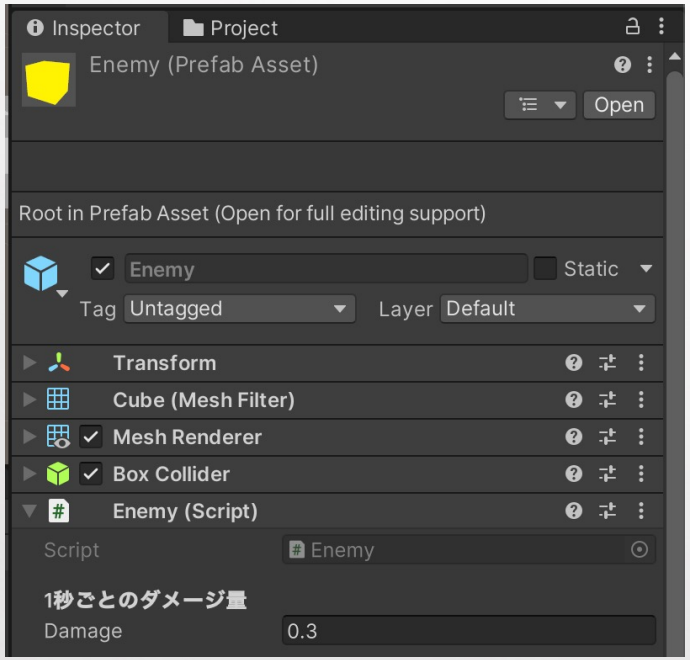
２．ゲームの作成

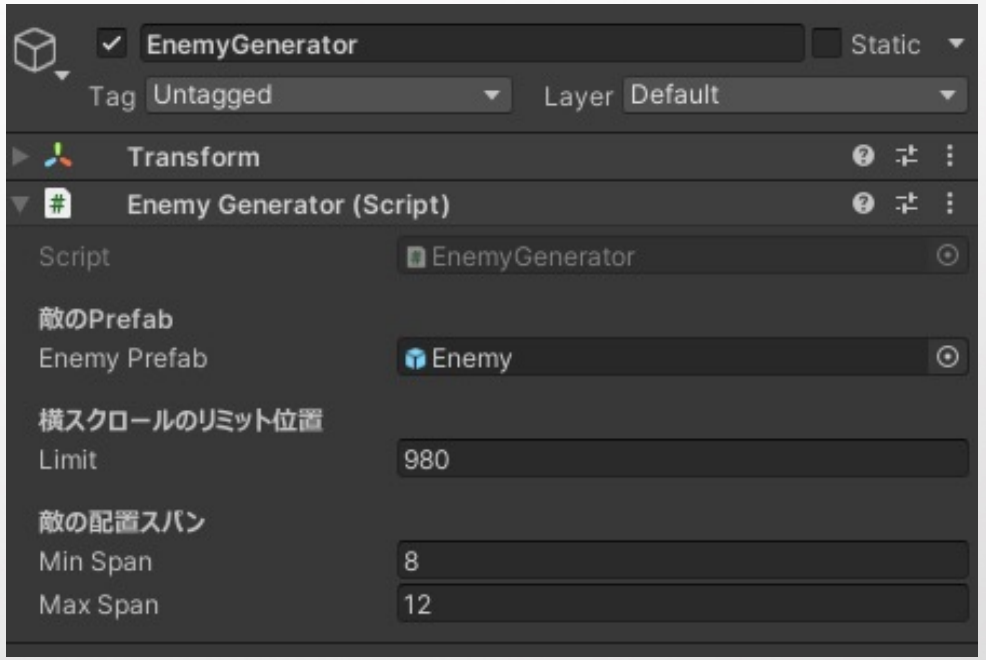
＜プログラムの作成＞

１．Enemy.csを作成し、作成後Enemyにアタッチする。

２．EnemyGenerator.csを作成し、空のGameObjectを作成しアタッチ。

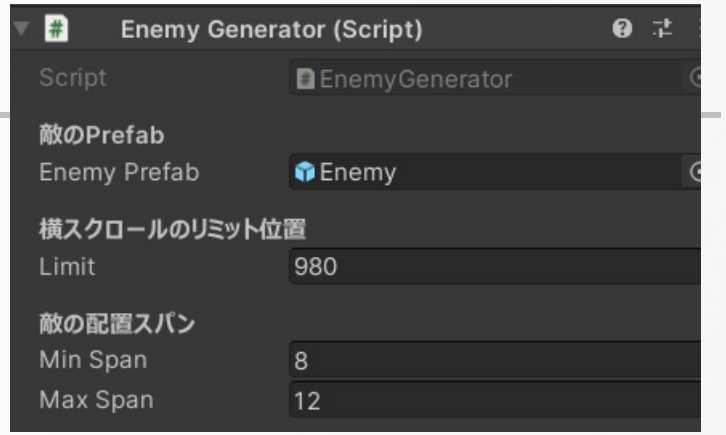
　　１と２のUnity上の設定値はとりあえず以下のように設定。





まずは、ここまでで再生し、Field上に黄色のEnemyが配置されることを確認。

２．ゲームの作成



＜プログラム解説＞

EnemyGenerator.cs

[Header(“xxxxxx”)]

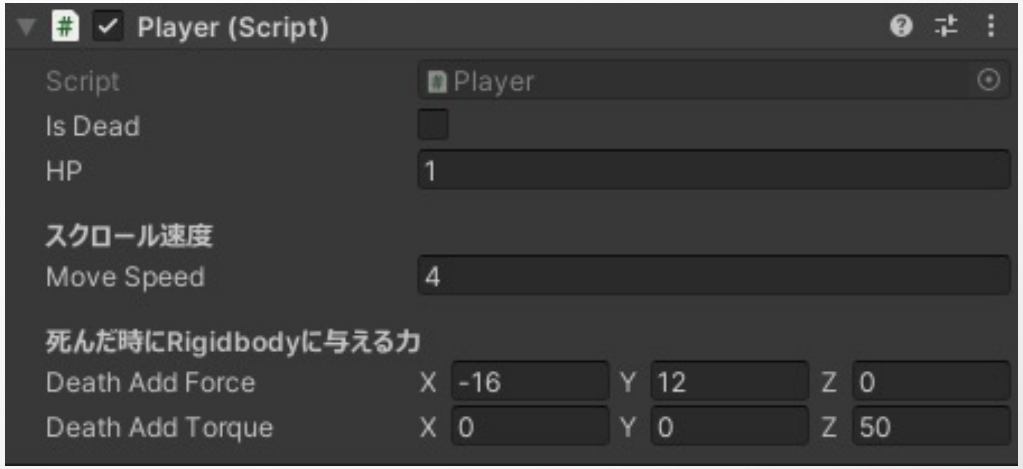
上記の宣言をすると、Unity上のインスペクターに説明が明記される。

日本語OKなので、チーム開発の時などに使用すると親切！

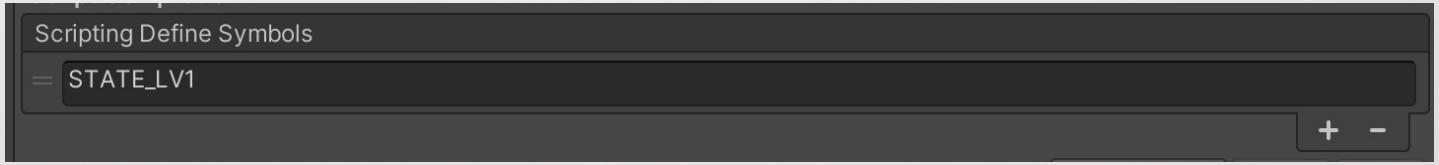
処理内容としては、(10,0,0)の位置からminSpanとmaxSpanの座標分あけて空中か地面に敵を繰り返し配置するというプログラム。

２．ゲームの作成

３．次にPlayerのスクリプトを作成する。今までの方法とは少し異なるので、注意。(Player.cs / Player_Common.cs)
Unity上の設定は以下の通り。



４．BuildSetting⇨PlayerSettingからScriptingDefineSymbolsに以下の項目を追加。



２．ゲームの作成

５．CameraMove.csを作成後、カメラにアタッチ。
　　同様に、HPGauge.csを作成後、HPGaugeにアタッチ。

２．ゲームの作成

＜プログラム解説＞

Player_Common.cs/Player.cs

partial

スクリプト(ファイル)を分けて同じスクリプトを記述することが出来る宣言。

実態は1つのクラス内に記載しているのとまったく同じ。

rb.AddTorque

回転する力を加える

今後、Player_Common.csのJump(ジャンプ)、Squat(しゃがみ)、EndSquat(立つ)、Dive(ダイブ)の関数を使用するため、内容確認してみてください。

２．ゲームの作成

＜プログラム解説＞

・条件付きコンパイル

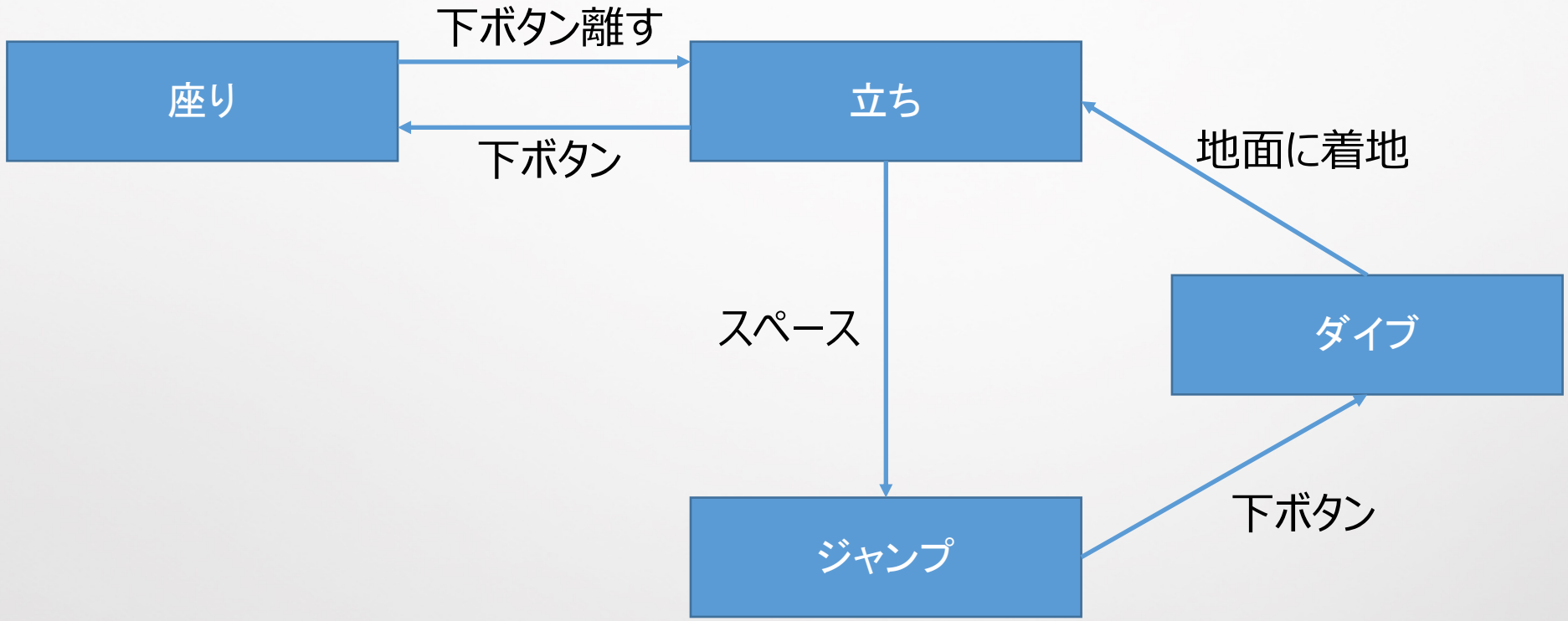
#if ～ #endifを使用することで、特定のdefineが定義されているときだけ処理を行うような記述が可能。defineはスクリプトの中で定義することも可能ですがやってもらったように、BuildSetting内で実装することもできます！

以下のように、開発中のときのみDebug.Logを出力することとかも可能なので、ぜひ活用してみてください。

```
#if DEBUG
        Debug.Log("DEBUGというdefineがあるときだけ出力");
#endif
```

２．ゲームの作成

＜プレイヤーのステートのまとめ(状態遷移図)＞



３．ステートパターンLv1(enum)

＜Lv1：enumでの管理＞

よくある実装として、ステートをenumで表すことが多いと思います。
個人開発や小規模なプロジェクトであれば、これで十分だと思います。

ただ、状態ごとの処理や振る舞いについてはif文やswitch文を用いて
enumの値に応じて分岐処理が必要になってくると思います。
今後ステータスがどんどん追加されるようなゲームであれば、どんどん見にくく
なっていくことが懸念されます。

では、実際にステートパターンを実装してみましょう！Player.csと付随する
ステータスのクラスを作成して、STATE_LV2に設定してください。

４．ステートパターンLv2(クラス)

＜Lv2：クラスでの管理＞

中規模、大規模なゲーム開発ではクラスを使ってステートパターンを作る場合が一般的。

ステートごとに以下のメソッドを保持する。

・ステートを開始した場合に呼ばれるメソッド

・毎フレーム呼ばれるメソッド

・ステート終了時に呼ばれるメソッド

例：ジャンプのステート

・ステートを開始した場合に呼ばれるメソッド　⇒　ジャンプ処理

・毎フレーム呼ばれるメソッド　⇒　下押されていたら、ダイブに切り替え

・ステート終了時に呼ばれるメソッド　⇒　処理なし

４．ステートパターンLv2(クラス)

＜Lv2：クラスでの管理＞

メリットとしては、どれだけステートが追加されてもメソッドとしては3つの
観点のみ抑えれば問題なし。(ステートがパターン化されている)
仕様書も書きやすくなる。マトリクス図を穴埋めしていくだけで完成出来る。

デメリットとしては、ステートパターンを知らない人から見た場合、まずはこの
概念を覚えることが必要となる。
また、ステートが2,3パターンの場合はenumで見た方が早い場合も。

出来た人はステートパターンを使って、Dキーを押した時にダッシュする
ステータスを実装してみてください！