Unityはじめるよ
～基礎３～
～応用１～

統合開発環境を内蔵したゲームエンジン
http://japan.unity3d.com/

※いろんな職業の方が見る資料なので説明を簡単にしてある部分があります。正確には本来の意味と違いますが上記理由のためです。ご了承ください。
この資料内の一部の画像、一部の文章はUnity公式サイトから引用しています。

本日勉強する内容

基礎３：Prefab、コライダー
応用１：uGUI概要

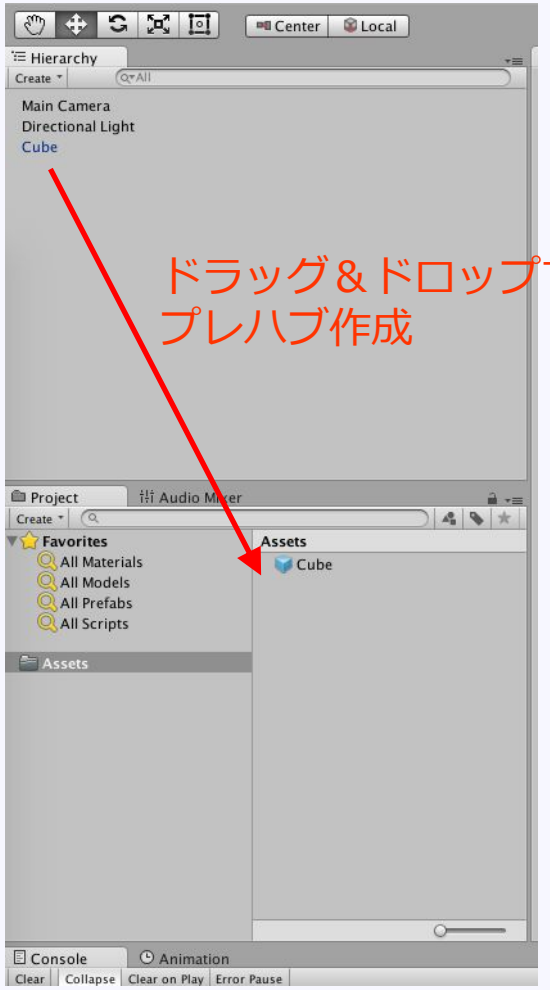
■Prefab（プレハブ）

・**プレハブとは？**
再利用可能なGameObjectのこと。

って言われてもよくわからないよね。
簡単に言うと、
ヒエラルキービューで作ったGameObjectを、
プロジェクトビュー側に保存したモノのこと。

ヒエラルキービューからプロジェクトビューに
ドラッグ＆ドロップすることでプレハブ作成。

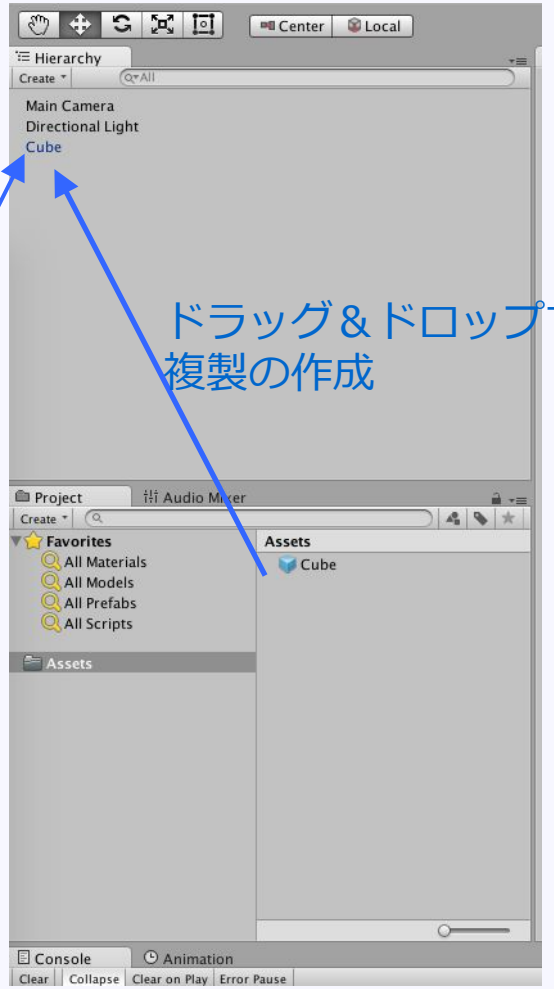
プロジェクトビュー側に保存しておくことで、
複製（クローン）を作る元として利用出来る。



■プレハブを作る方法

プレハブをヒエラルキービュー側に
ドラッグ＆ドロップすることで、
複製を実体化（**インスタンス化**）できる。

ちなみに複製したものは、
ヒエラルキービュー上では青文字で表示される。



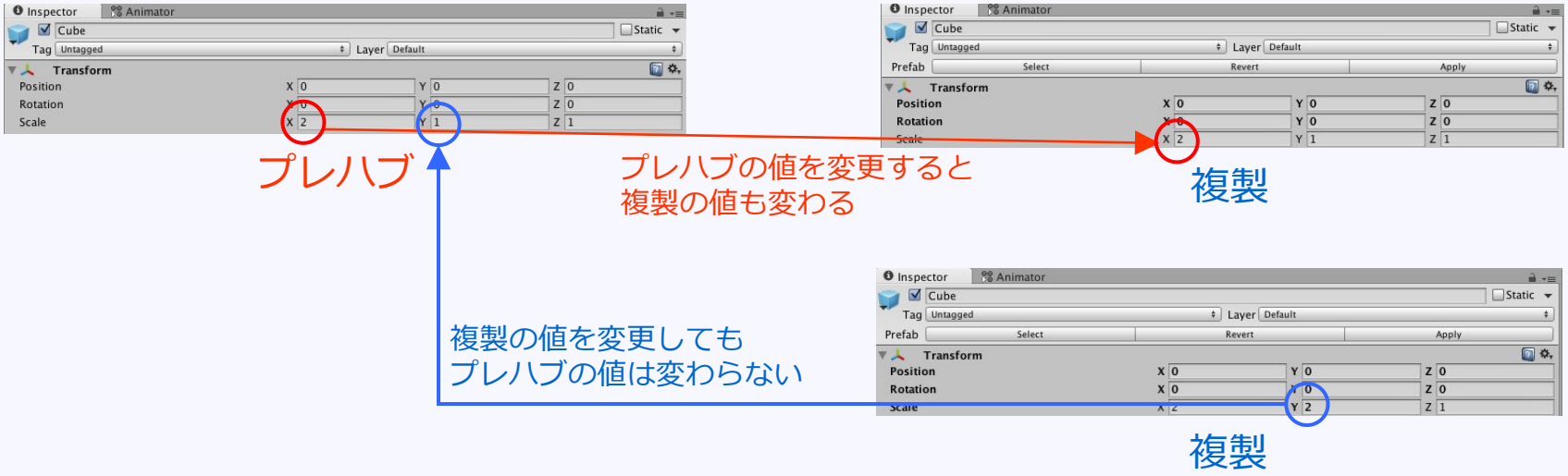
「ヒエラルキービュー上だけでもGameObject
を複製できるじゃん」って思った人。
そう、その通り。
ヒエラルキービュー上でも簡単に複製は作れる。
ただし、
それとプレハブでは決定的な違いがある。

■継承(Inheritance)

プレハブからの複製の場合は、オリジナルに手を加えると、ヒエラルキービュー上の複製全てに反映されるという特徴がある。

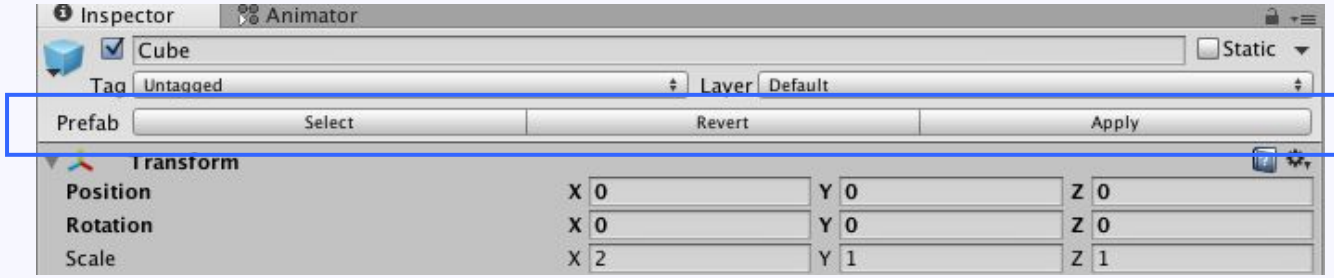
これを**「継承(Inheritance)」**という。

オリジナルと複製がリンクしているということ。
オリジナルをいじると複製に反映されるが、複製をいじってもオリジナルには反映されない。



■複製のインスペクタビュー上のボタン

複製をインスペクタビューで見てみると、上部に
「Select」「Revert」「Apply」というボタンがあることに気づく。



それぞれのボタンの意味は以下の通り。

Select
　どのプレハブからの複製なのかをプロジェクトビュー上に表示する

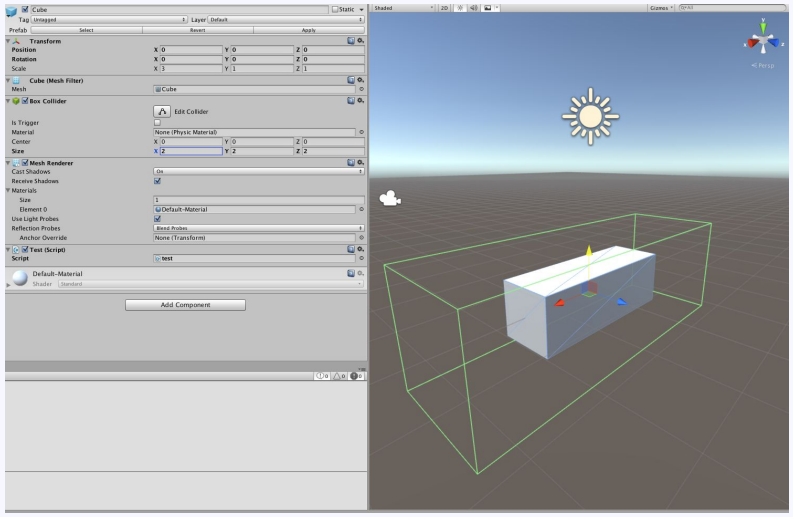
Revert
　複製側に加えた変更をリセットし、オリジナルと同じ状態に戻す

Apply
　複製側に加えた変更を、オリジナルに反映する

上部メニューのGameObject > Break Prefab Instanceで、リンクを解除することもできる。

■コライダー

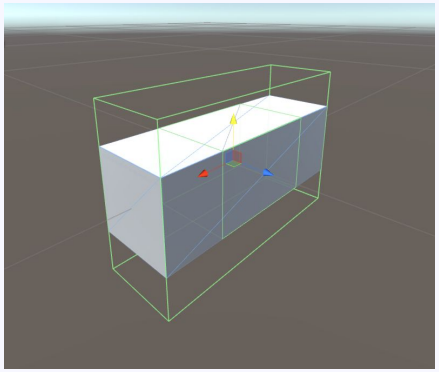
・**コライダーとは？**
衝突判定（当たり判定）を行うコンポーネントのこと。
シーンビュー上では緑の線で表示される。
インスペクタービュー上で設定する値は、
Transformコンポーネントの値からの相対的な値となる。

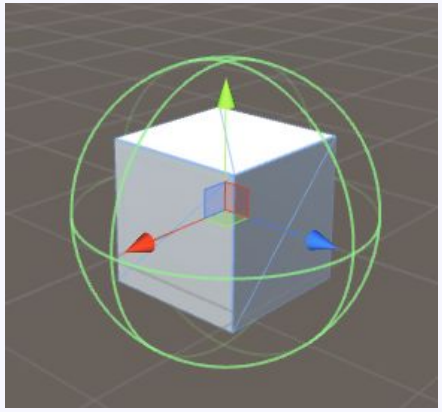


※参考Unity公式マニュアル
http://docs.unity3d.com/ja/current/Manual/CollidersOverview.html

■コライダーの形状

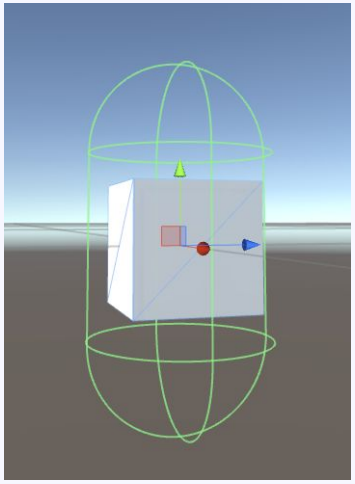
・ボックス・・・立方体



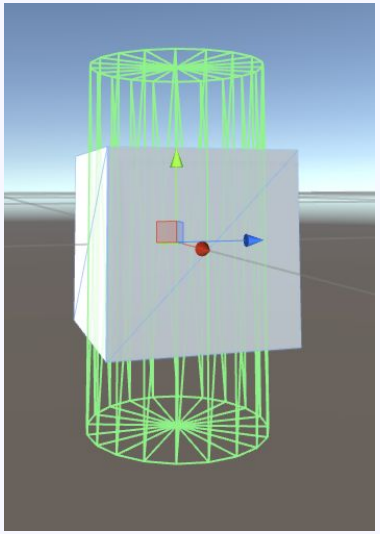


・スフィア・・・球状

・カプセル・・・カプセル形状



■コライダーの形状



・メッシュ・・・Meshを選択して衝突判定を生成

・ホイール・・・車のタイヤ用（かなり細かい設定項目がある）
・テレイン・・・Terrainデータを選択して地形用衝突判定を生成

これらを組み合わせて、最適な当たり判定を作る。
人型モデルだったら、カプセル形状が向いている。
メッシュコライダーは複雑な衝突判定を作れるが処理が重たい。
モバイルで利用するなら、ボックス、スフィア、カプセルあたりを
組み合わせて使った方が良い。

■コライダーの種類

・静的コライダー
　物理演算特性を持たないGameObject向け。
　地形や建物などの動かないGameObjectに利用する。
　Rigidbodyコライダーとは衝突判定を行うが、
　静的コライダー同士は衝突判定を行わない。

・Rigidbodyコライダー
　物理特性を持たせるGameObject向け。
　他のコライダーと衝突判定が行われ、
　物理演算の挙動に反映される。
　物理演算を利用するゲームでは一番よく使われるこライダー。

・Kinematic Rigidbodyコライダー
　物理演算の影響は受けないが、動かせるGameObject向け。
　他のコライダーとの衝突判定は行われる。
　ドアなど、特定の条件の時のみ動くGameObject向け。

■トリガーコライダーと非トリガーコライダー
インスペクタービューのisTriggerのチェックを入れると
トリガーコライダーとなる。

・トリガーコライダー
衝突の検出のみを行うために利用。（イベント駆動用）

・非トリガーコライダー
衝突判定と同時に物理演算を行う。（プレイヤー、障害物など）

■レイヤー
コライダーはGameObjectが属するレイヤー間の
衝突判定の制御が可能。

上部メニューEdit > Project Settings > Physicsから
レイヤー間での衝突判定のON/OFFを設定できる。

■スクリプトとの連携

コライダーの付いたGameObjectは、スクリプトで衝突判定を受け取ることができる。

主要なオーバーライド関数（メッセージ）は以下の通り

OnCollisionEnter
この collider/rigidbody は他の collider/rigidbody に触れた時に OnCollisionEnter は呼び出されます。

OnCollisionExit
この collider/rigidbody が他の collider/rigidbody と触れ合うのをやめた時に OnCollisionExit は呼び出されます。

OnCollisionStay
OnCollisionStay は rigidbody/collider が他の rigidbody/collider に触れている間毎フレーム 1 度だけ呼び出されます。

OnTriggerEnter
Collider が他のトリガーイベントに侵入した時に OnTriggerEnter が呼び出されます。

OnTriggerExit
Collider が other のトリガーに触れるのをやめた時に OnTriggerExit は呼び出されます。

OnTriggerStay
OnTriggerStay はトリガーが 他の Collider に触れ続けている間 ほとんど 毎フレーム呼び出されます。

■OnCollision系（非トリガーコライダー）
引数でCollisionクラスが渡される。Collisionクラスには接触点、衝突した速度などの情報が含まれている。

```
void OnCollisionEnter(Collision collision)
{
    foreach(ContactPoint point in collision.contacts)
    {
            // 衝突地点の取得
        Vector3 hitPos = point.point;
    }
}
```

■OnTrigger系（トリガーコライダー）
引数でColliderが渡される。

```
void OnTriggerEnter(Collider other)
{
    // 大体の衝突地点の取得
    Vector3 hitPos = other.ClosestPointOnBounds(this.transform.position);
}
```

※参考Unity公式マニュアル
http://docs.unity3d.com/ja/current/ScriptReference/Collider.html

■uGUI（概要）

・**uGUIとは？**

uGUIとはUnity純正のGUI作成機能。

※GUIはGraphical User Interfaceの略

簡単に言うとメニュー表示などに利用するパーツを扱う仕組み。

uGUIという仕組みができる前までは、
プログラムからGUIの表示を行っていた。
※それが大変なので、みんなNGUIという便利なアセットを使っていた

uGUIでは、エディタ上で簡単にGUIを扱えるようになった。
もちろん、スクリプトとの連携も可能。

■Canvas

・Canvasとは
UIパーツを配置する親GameObjectで、
UIパーツは必ずCanvas配下に追加する必要がある。
Canvasは複数作成することが可能。

・Canvasの作成
上部のメニューGameObject > UI > Canvas
でキャンバスを作成。
※このときEventSystemというGameObjectも一緒に追加される

■RectTransform

・RectTransformとは
uGUI用に作られたTransformを拡張したコンポーネント。
Transformの位置・回転・スケールに加え、
サイズ・アンカー・ピボットという概念が追加されている。
これらはお互いが深く作用しあうことになる。

サイズ(Size)
UIの大きさを表す。

アンカー(Anchor)
親のRectTransformからの比率で、
表示位置やサイズを設定する仕組み。

ピボット(Pivot)
回転や拡大縮小の中心点。
ピボットの位置はUIを0~1の正規化した値で表す。
左下が(0,0)で右上が(1,1)

■RectTransformの設定項目

・**Render Mode**
UI をスクリーンに描画するか、3D 空間のオブジェクトとして存在させるかの方式。

Screen Space - Overlay
完全に2Dで描画。
ディスプレイの解像度やアスペクト比が基準となる。

Screen Space - Camera
その他のGameObjectと前後関係がある。
ディスプレイの解像度やアスペクト比が基準となる。

World Space
3D空間でのUI表現。
ディスプレイの解像度やアスペクト比は利用せず、
RectTransformの値を利用する。

3Dオブジェクトとの干渉が必要ないなら、Screen Space - Overlay
3Dオブジェクトとの干渉が必要なら、Screen Space - Camera
3D空間上にUIを配置したいなら、World Space
と使い分けるのが良い。

【Screen Space 系の場合の設定項目】
・**Pixel Perfect**
精度維持のために UI をアンチエイリアス無しで描画するか。

【Screen Space - Overlayの場合の設定項目】
・**Sort Order**
複数のCanvasが存在する場合：数字が大きいほど前に表示される
同じ値の場合は描画順は不定となる。

【Screen Space - Cameraの場合】
・**Render Camera**
UI を描画するカメラ（詳細は下記）。

・**Plane Distance**
UI の平面が描画担当のカメラの前からどのくらい離れて配置されるか。

・**Sorting Layer**
複数の2Dオブジェクトが存在する場合は、数字が大きいほど前に表示される

・**Order in Layer**
複数の2Dオブジェクトが存在し、同じレイヤーを利用している場合：数字が大きほど前に表示される。
Sorting LayerもOrder in Layerも同じ場合はカメラから近いほど手前に描画される。

【World Spaceの場合】

・**Event Camera**
UI イベントを処理するために使用されるカメラ。

・**Sorting Layer**
複数の2Dオブジェクトが存在する場合：数字が大きいほど前に表示される

・**Order in Layer**
複数の2Dオブジェクトが存在し、同じレイヤーを利用している場合：数字が大きほど前に表示される

Sorting LayerもOrder in Layerも同じ場合は、カメラから近いほど手前に描画される。

・**Receives Events UI**
イベントがこの Canvas 処理されているかを確認するための機能。

※補足　Canvas内での表示順序について
Canvas内のオブジェクトについては、ヒエラルキービュー上での順番通りに表示される（下にあるものほど手前）
スクリプトからも調整可能
this.transform.SetSiblingIndex(1);　　// 数字が大きものほど手前

■**CanvasScaler**
画面解像度やアスペクト比が違う端末での表示方法を決める項目

・**Constant Pixel Size**
UI 要素をスクリーンサイズによらずピクセル単位で同様のサイズに保つ。

・**Scale With Screen Size**
画面サイズが大きいほどに、UI 要素を大きくする。

・**Constant Physical Size**
UI 要素をスクリーンサイズや解像度によらず、物理的に同様のサイズに保つ。

■**Graphics RaycasterScalerの設定項目**
タップ位置にどのUIパーツがあるかを判定する仕組み

ご清聴ありがとうございました