CHAPTER4_アセット管理とゲームオブジェクトの制御 ー プレハブとエフェクトを極める

画像（テクスチャ）を扱う方法、プレハブ、パーティクルについて学ぶ。

Chapter4 の目的

- 画像や3Dモデル等の外部アセットを利用できるようになる
- プレハブを利用してオブジェクトを動的に生成できるようになる
- パーティクルシステムやサウンドを使えるようになる

4-1　アセットのインポート

アセットをひとまとめにしたUnityパッケージを利用すると、手間なくアセットをインポートできる。インポートしたパッケージはプロジェクトビューに表示される。手順は以下の通り。

アセット　＞　パッケージをインポート　＞　カスタムパッケージ　＞　対象のパッケージを選択

インポート - Unity マニュアル
アセットパッケージ - Unity マニュアル

COLUMN　Import Settingsでのアセットの設定

インポートしたアセットを選択すると、インスペクター上から**インポート設定**することができる。設定はファイルの形式によって異なる。インポートした時点で（エクスポート元と同じ）設定が行われており、プラットフォームを変更する、画像のサイズを変更して軽くする等の特別な目的が無い限り、編集する必要はない。講義ではインポート設定を変更しない。

4-2　ステージの作成

ステージの組み立てとコライダーの編集

（特記事項なし）

テクスチャーの利用

テクスチャはマテリアルのアルベドに設定することができる。タイリングは入力した数値分だけテクスチャを繰り返し並べる。

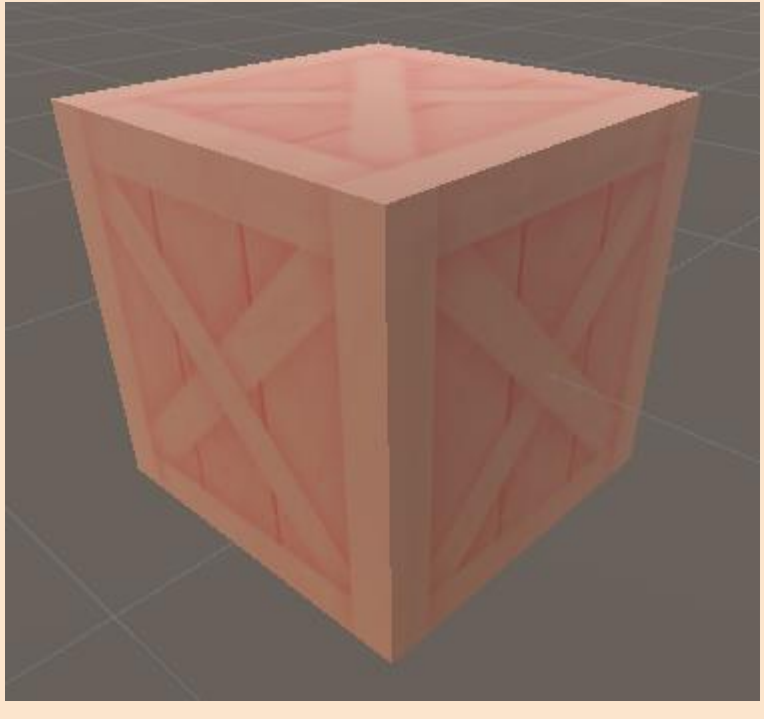
COLUMN　マテリアルのノーマルマップ

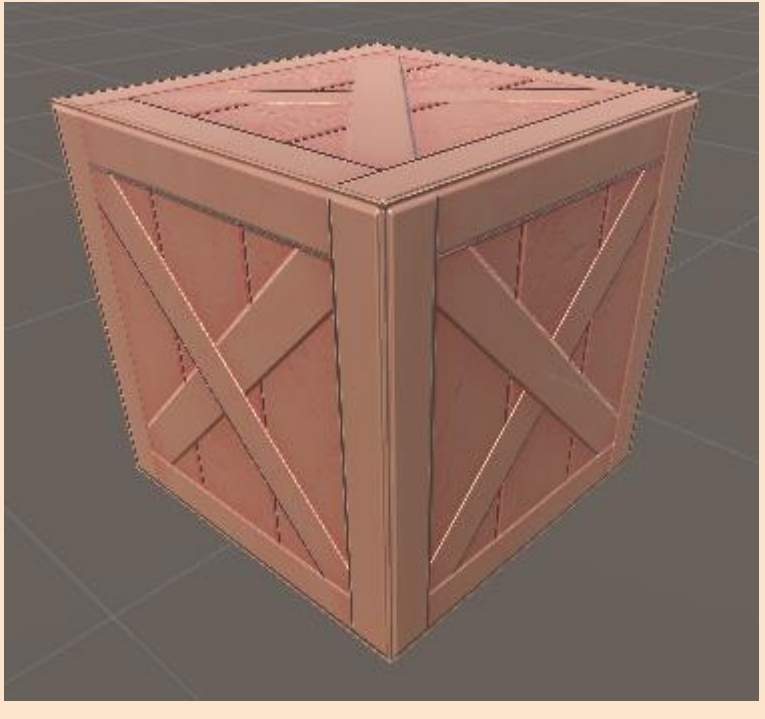
マテリアルに**ノーマルマップ（法線マップ）**を設定すると、表面の凹凸を表現することができる。法線マップはテクスチャとセットで配布されている場合が多く、その場合は忘れずに法線マップを設定しておくこと。

法線マップ（Normal Map）（Bump mapping） - Unity マニュアル

図4-1　法線マップなし

図4-2　法線マップあり





ステージのデコレーション：3Dモデルの利用

プロジェクトビュー中の3Dモデル（教科書ではFBX形式）はヒエラルキー　もしくは　シーンにドラッグ＆ドロップすると、配置することができる。3Dモデルは通常テクスチャ等の設定を事前に済ませてあり、配置するだけですぐ使える状態になっていることが多い。設定でテクスチャを使うことにしてあるのであれば、該当テクスチャが用意されているハズ。

図4-3　配置した3Dモデル（マテリアル設定済み）

インスペクター

Can

Shader Standard

Rendering Mode Opaque

Main Maps

⊙アルベド

⊙メタリック

スムースネス 0.591

ソース Metallic Alpha

⊙法線マップ

⊙ハイトマップ

⊙オクルージョン

⊙詳細マスク

放出

タイリング X 1 Y 1

オフセット X 0 Y 0

ステージのデコレーション：CANDY DOZERロゴの配置

PlaneとQuad

Plane（平面）・Quad（クアッド）は共に平面を表す。PlaneはデフォルトでXZ軸で使い、QuadはXY軸で使う。Plane・Quadはメッシュのポリゴン数が異なり、Planeの方が圧倒的に多い。特別な事情が無い限り、Quadを使うと良い。

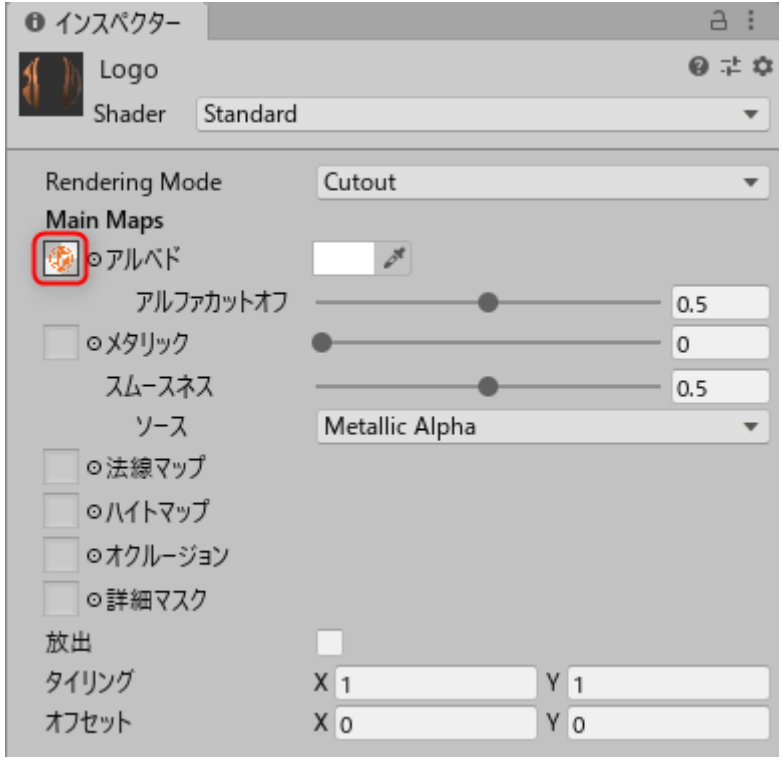
プリミティブとプレースホルダーオブジェクト - Unity マニュアル
メッシュ - Unity マニュアル

マテリアルでテクスチャを指定

マテリアルのアルベドは色だけでなく、テクスチャーを指定することができる。テクスチャーの指定は以下2つの方法がある。

- 下図の赤線で囲った部分にテクスチャーをドラッグ＆ドロップする
- 線で囲った部分の右の○をクリックしてテクスチャーを選択する

図4-4　マテリアルのアルベド（テクスチャー指定）


インスペクター
Logo
Shader
Standard
Rendering Mode
Cutout
Main Maps
アルベド
アルファカットオフ
0.5
メタリック
0
スムースネス
0.5
ソース
Metallic Alpha
法線マップ
ハイトマップ
オクルージョン
詳細マスク
放出
タイリング
X 1
Y 1
オフセット
X 0
Y 0

マテリアルのRendering Mode

オブジェクトが透明度（アルファ値）を使うかどうかを選択する。デフォルト設定の**Opaque**は透明度を使わずに完全に不透明。**Cutout**はアルファカットオフで指定された値未満のアルファ値部分を完全に透明にする。Cutoutはアルファ値が設定されているテクスチャを透過したい場合に使う。

Rendering Mode - Unity マニュアル
Albedo カラーと Transparency（透明度） - Unity マニュアル

図4-5　Rendering Modeの指定（Cutoutでアルファカットオフを設定の場合）

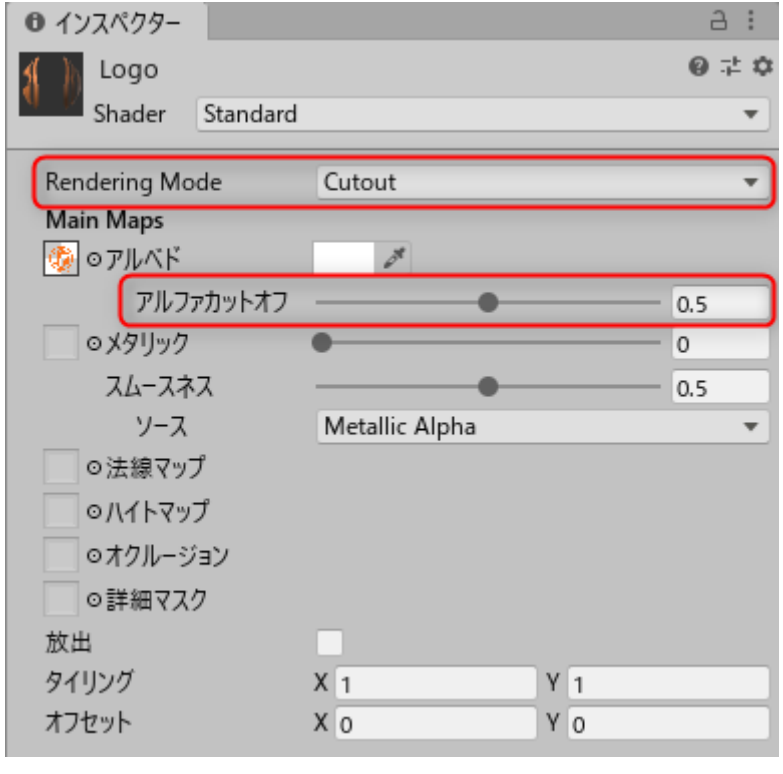

インスペクター
Logo
Shader
Standard
Rendering Mode
Cutout
Main Maps
アルベド
アルファカットオフ
0.5
メタリック
0
スムースネス
0.5
ソース
Metallic Alpha
法線マップ
ハイトマップ
オクルージョン
詳細マスク
放出
タイリング
X 1
Y 1
オフセット
X 0
Y 0

図4-6　Rendering Mode：Opaque

図4-7　Rendering Mode：Cutout

その他のステージのデコレーション：連続旗の作成

（特記事項なし）

その他のステージのデコレーション：クロスの作成

コンポーネントを削除

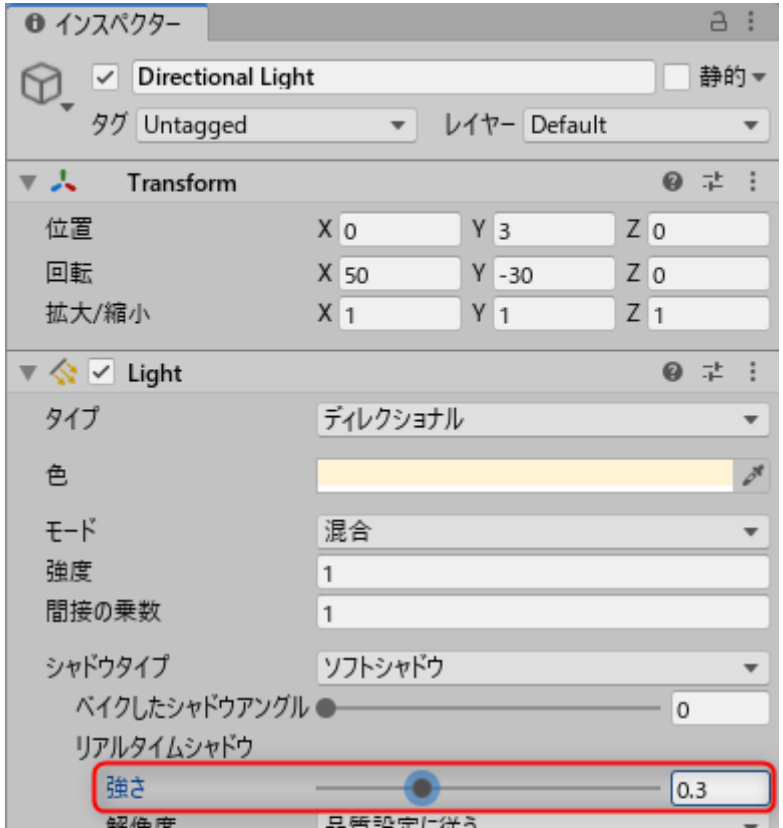
インスペクター上からコンポーネントを削除できる。削除したいコンポーネントの右上にあるメニューアイコンから削除可能。

カメラとライトの調整

影の強さ調整

ディレクショナルライトの設定の強さで影の強さを調整することができる。

図4-8　影の強さ調整


インスペクター
Directional Light
静的
タグ Untagged
レイヤー Default
Transform
位置 X 0 Y 3 Z 0
回転 X 50 Y -30 Z 0
拡大/縮小 X 1 Y 1 Z 1
Light
タイプ ディレクショナル
色
モード 混合
強度 1
間接の乗数 1
シャドウタイプ ソフトシャドウ
ベイクしたシャドウアングル 0
リアルタイムシャドウ
強さ 0.3

プッシャーの作成

ごまかす

3Dオブジェクトは複雑な形状であればあるほどポリゴン数が増えて描画に負荷がかかる。また、複雑な3Dオブジェクトを用意するのは手間がかかる（＝コストがかかる、時間がかかる、お金がかかる）ので、さほど重要でない等でごまかせるところはテクスチャ等で積極的にごまかすと良い。

プッシャーの制御

Mathf.Sinメソッド

Mathf.Sinメソッドは三角関数のサイン（Sign）を計算する。とてもザックリ言うと、Mathf.Sin(x)はxの増減によって-1.0～1.0の範囲を行ったり来たりする。この範囲を大きくするならば、Mathf.Sin(x)の結果を何倍かすればよい。xの増減が大きいほど、より早く行ったり来たりする。

例8-1　-5.0~5.0の範囲を行ったり来たりする

```
Mathf.Sin(x) * 5;
```

例8-2　2倍の早さで行ったり来たりする

```
Mathf.Sin(x * 2);
```

Mathf-Sin - Unity スクリプトリファレンス

…ちなみに、Mathf.Sinメソッドの引数はラジアンである。引数の単純な角度を求めたい場合は(180 / Mathf.PI)を掛けてやればいい。

例8-3　xの角度を求める

```
x * 180 / Mathf.PI;
```

ラジアン - Wikipedia

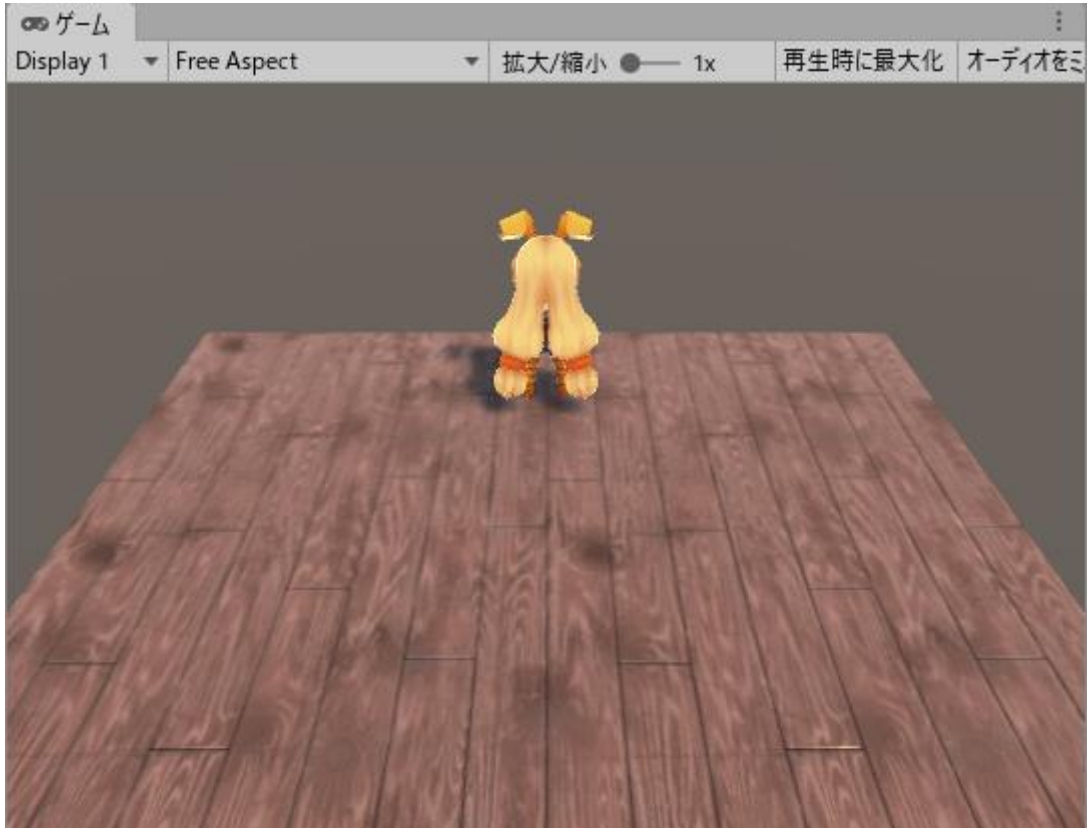
Time.time

Timeクラスは時間情報を取得するためのクラス。様々なpublicでstaticなプロパティが用意されており、ゲームでは頻繁に使うだろう。

Time.time	ゲーム開始からの時間。読み取り専用プロパティ、戻り値の型はfloat。
Time.deltaTime	前フレームからの時間。読み取り専用プロパティ、戻り値の型はfloat。

Timeクラス (https://docs.unity3d.com/ja/2018.4/ScriptReference/Time.html)
Time-time - Unity スクリプトリファレンス

図4-9　Time.deltaTimeを利用した例（キャラクター位置、回転をスクリプトで時間変化量によって制御）


ゲーム
Display 1
Free Aspect
拡大/縮小
1x
再生時に最大化

COLUMN　Transformのpositionプロパティの変更

transform.position.x = 10f;とするとエラー
Vector3は構造体であり、Transform.positionは戻り値の型がVector3のプロパティ。構造体が戻り値のプロパティのメンバ変数に値を代入するとエラーになる。取得はエラーにならない。
構造体は値型なので、Transform.positionは参照情報ではなく値が戻り値となる。参照情報のない構造体のメンバ変数に値を代入しても意味はないのでエラーになる仕様になっているようだ。

参照と実体 - プロなら当然！プログラミング技能解説 #2　※8:00くらいから見ると良い

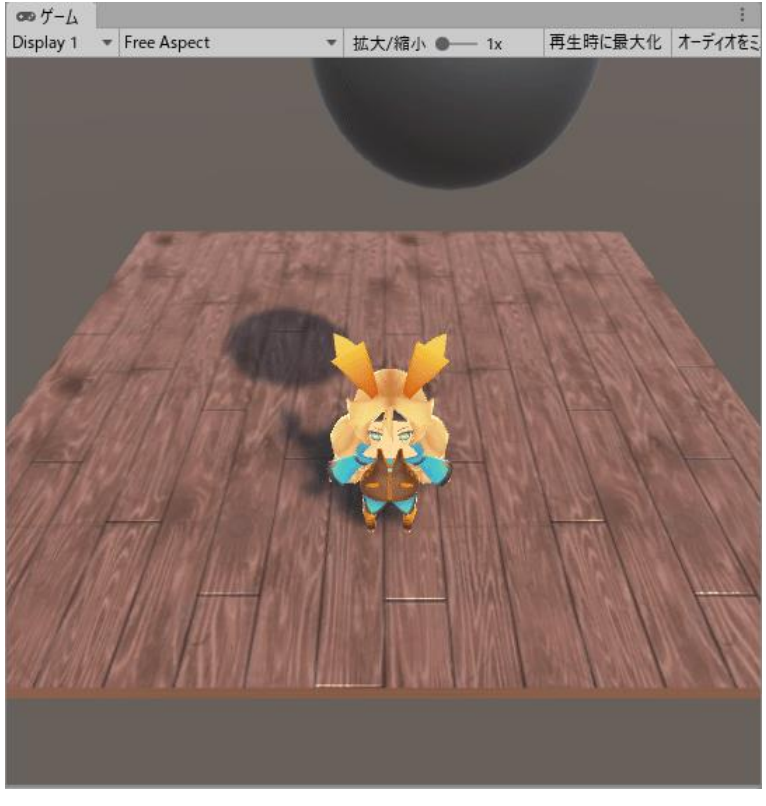
Rigidbodyのis Kinematic

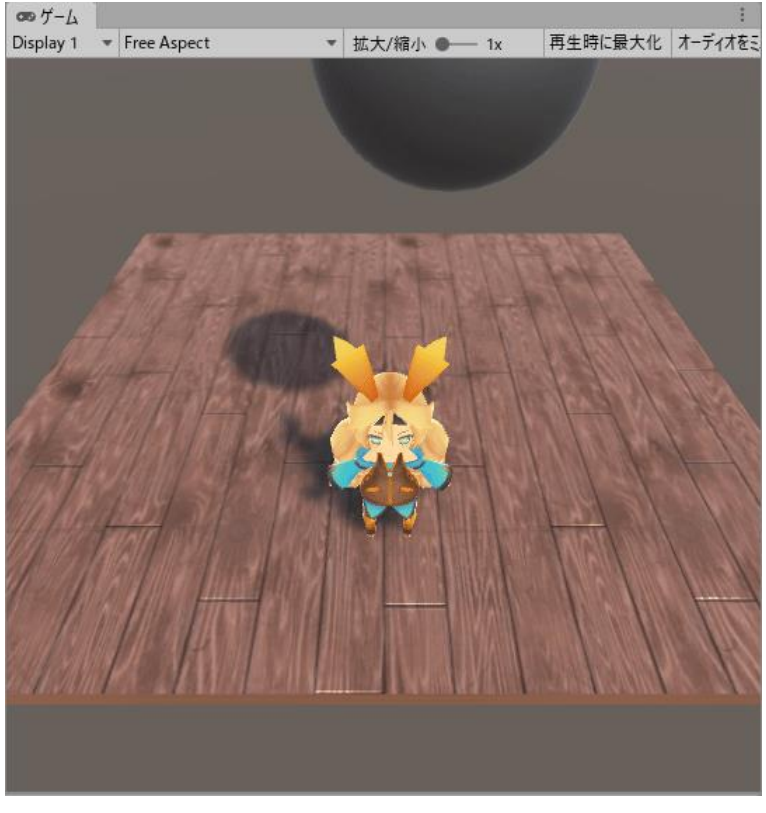
Rigidbodyのパラメータのis Kinematicにチェックを入れると、物理的な影響を受けなくなる。AddForce等のメソッドでも動かなくなるため、移動させるにはスクリプトから位置を変更する必要がある。

Rigidbody - Unity マニュアル

図4-10　ユニティちゃんのis Kinematic：**無効**

図4-11　ユニティちゃんのis Kinematic：**有効**


ゲーム
Display 1
Free Aspect
拡大/縮小
1x
再生時に最大化
オーディオをミ


ゲーム
Display 1
Free Aspect
拡大/縮小
1x
再生時に最大化
オーディオをミ

4-3　プレハブの作成と利用

プレハブとは

プレハブは関連性のあるアセットをひとまとめにしたアセット。フォルダと違って、プレハブ自体がアセットと見なされる。テンプレートとして使うことができ、プレハブ1つから複数の複製（クローン）を作ることが多い。プレハブの内容を編集すると、複製先にも適用される。複製先で設定値をオーバーライドすることも可能。

プレハブ - Unity マニュアル

Candyオブジェクトの生成

`Mesh Collider`

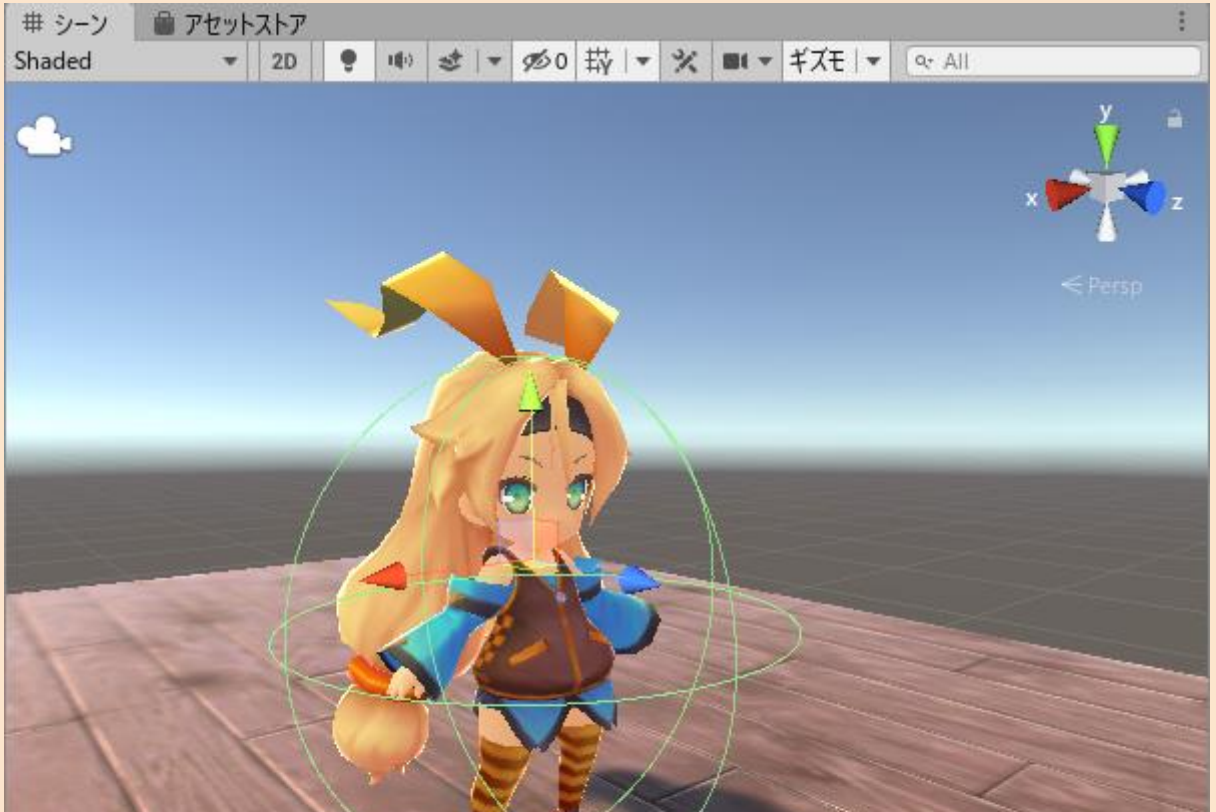
メッシュコライダー（Mesh Collider）はメッシュに合わせる形で形成されるコライダー。凸状にするにチェックを入れるとメッシュコライダー同士に衝突判定が行われる。

Mesh Collider - Unity マニュアル

COLUMN　Mesh Colliderと複雑な形状への対応

メッシュコライダーはポリゴンの三角形の面の数に（255までの）制限があり、複雑なメッシュの場合はコライダーが簡略化される。複雑なコライダーが必要なケースは少なく、大抵はおおまかなコライダーでなんとかなる。

図4-12　おおまかなコライダー（カプセルコライダー）


シーン
アセットストア
Shaded
2D
ギズモ
All
y
x
z
Persp

CandyOrangeオブジェクトのプレハブ化と利用

プレハブ化

ヒエラルキーからプロジェクトビューにオブジェクトをドラッグ＆ドロップすると、**プレハブを作成**することができる。対象のオブジェクトがプレハブだった場合、プレハブ化する際に**原型プレハブ**か**プレハブバリアント**か選ぶことができる。プレハブ化したアセットはヒエラルキーもしくはシーンにドラッグ＆ドロップでいくつでも配置できる。

原型プレハブ

原型プレハブ（Original Prefab）は自身をベースとする状態でプレハブ化する。

プレハブバリアント

プレハブバリアント（Prefab Variant）は元のプレハブをベースとする状態でプレハブ化する。オーバーライドしてプレハブを使う場合にはプレハブバリアントを使うと便利。プログラム的に表現するならば、プレハブ元を継承して必要なメンバをオーバーライドする感じ。

プレハブの作成 - Unity マニュアル
プレハブバリアント - Unity マニュアル

他のキャンディの作成とプレハブモードによる編集

プレハブモード

プレハブをアクティブにした状態でインスペクター上の**プレハブを開く**ボタンをクリックすると**プレハブモード**で編集ができる。

TIPS　プレハブモードのAuto Save

プレハブモードの右上にある**自動保存**にチェックを入れると、自動的に編集内容が保存される。チェックを外した場合はCtrl + Sまたは画面右上の**保存ボタン**で保存することができる。

※未保存でプレハブモード終了時は保存するかどうか尋ねられる

COLUMN　プレハブの継承ができるプレハブバリアント

プレハブバリアントは管理が難しいので慣れた後で使うと良い。

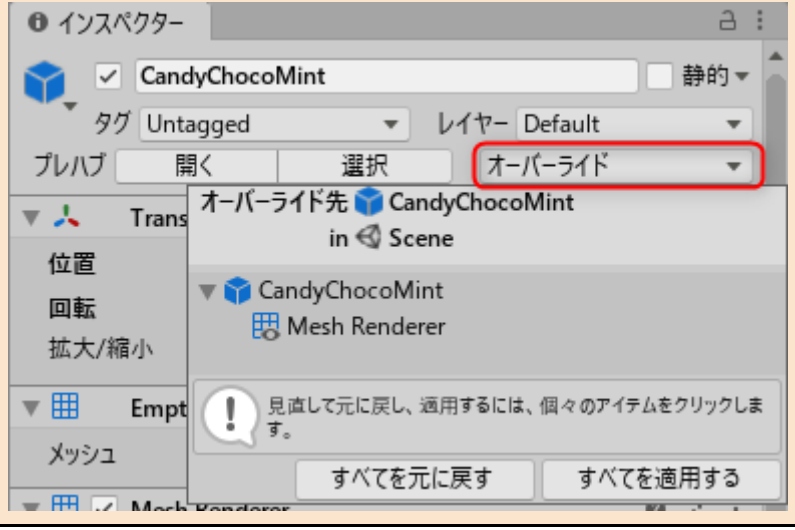
初期キャンディの配置

（特記事項なし）

COLUMN　プレハブ変更とシーンのオーバーライド

シーン配置したプレハブで設定をオーバーライドしている場合、元のプレハブを修正してもオーバーライドが優先される。オーバーライドした設定はインスペクタ上の**オーバーライドコンボボックス**から元のプレハブに適用することもできる。

図4-13　オーバーライドがある場合のオーバーライドコンボボックス



4-4　オブジェクトの動的生成と削除

Shooterの実装

プレハブはGameObject型

スクリプト上ではプレハブはGameObject型として扱う。インスペクタ上でパラメーターとする際にも同じ。

Input.GetButtonDownメソッド

引数で指定したボタンが押されている間だけtrueを返すメソッド。

Input-GetButtonDown - Unity スクリプトリファレンス

Instantiateメソッド

オブジェクトを動的に生成するには**Instantiateメソッド**を使う。引数で回転なしを指定する場合は**Quaternion.identity**を使うと良い。Quaternion.identityはワールド座標もしくは親の座標に沿って全く回転していない状態を表す。回転ありは後述する。

Instantiateメソッドで生み出されたオブジェクトは名前の末尾に**(Clone)**が付く。

ゲームオブジェクトの作成および削除 - Unity マニュアル

Object-Instantiate - Unity スクリプトリファレンス
Quaternion-identity - Unity スクリプトリファレンス

Quaternion

Unityでの回転は基本的に**Quaternion（クォータニオン）**で表す。Quaternionの概念は非常に難解で、一般的には3D空間の回転を表すものと理解しておけば問題ないが、メソッドの多くはQuaternionを必要とするので要注意。
インスペクター上のTransformでは直観的に操作できるように角度は**オイラー角**で表される。

Unity の回転と向き - Unity マニュアル

Rigidbody.AddTorqueメソッド

AddTorqueメソッドを使えばオブジェクトに回転する力を加えることができる。

Rigidbody-AddTorque - Unity スクリプトリファレンス

TIPS　MonoBehaviourのインスタンス化

MonoBehaviourを継承したクラスはnewでインスタンス化することができない。通常はInstantiateメソッドでオブジェクトを生み出す。

Shooterの改良

インスペクター上のパラメーターに配列を指定

publicな（もしくはSerializeFieldアトリビュートが付いた）メンバ変数に配列を使うことで、**インスペクター上のプロパティで配列を使うことができる**。オブジェクトやプレハブ等のアセットを値に設定する際は、インスペクター右上のロックアイコンでロックを掛け、対象のアセットを全て選択してからプロパティ名にドラッグ＆ドロップすると一度に設定できる。わざわざサイズ指定する必要が無いのでお勧め。

プロパティーの編集 - Unity マニュアル

図4-14　配列のプロパティ（ロックしてドラッグ＆ドロップすると楽）


インスペクター
Shooter
静的
タグ Untagged
レイヤー Default
Transform
位置 X 0 Y 2.5 Z -8
回転 X 300 Y 0 Z 0
拡大/縮小 X 1 Y 1 Z 1
Shooter (スクリプト)
スクリプト Shooter
Candy Prefabs
サイズ 4
要素0 CandyChocoMint
要素1 CandyChocoStrawberry
要素2 CandyLemon
要素3 CandyOrange
Candy Parent Transform なし (トランスフォーム)
Shot Force 560
Shot Torque 30
Base Width 0

Random.Rangeメソッド

最小値と最大値を指定してその範囲の乱数を発生させるRandom.Rangeメソッドがある。

Random-Range - Unity スクリプトリファレンス

親オブジェクトを設定する

Transformのプロパティparentにオブジェクトを代入すると、そのオブジェクトが親になる。

Transform-parent - Unity スクリプトリファレンス

Candyオブジェクトの削除

Destroyメソッド

Destroyメソッドを使えば引数のゲームオブジェクトを破棄できる。

Object-Destroy - Unity スクリプトリファレンス

TIPS　コンポーネントに対するDestroy関数の使用

Destroyメソッドはゲームオブジェクトだけではなく、コンポーネントを削除することも可能。しかし、通常はenabledを切り替えるだけで事足りる。

TIPS　タグの簡単な追加方法

インスペクター上からもタグを追加できる。

COLUMN　スクリプトリファレンス

今更感はあるが、スクリプトリファレンス　および　マニュアルは非常に役に立つので頻繁に見る癖をつけると良い。
Unity ユーザーマニュアル - Unity マニュアル
Unity スクリプトリファレンス

4-5　ゲームのコントロール

キャンディストックの消費

（特記事項なし）

キャンディストックの付与

（特記事項なし）

キャンディストックの自動回復

コルーチン

コルーチンはメソッドの一種で、StartCoroutineメソッドで呼び出されると、処理を一時終了して次のフレームで再開する仕組み。一時終了する際にはyield returnを用いる。戻り値の型はIEnumerator固定。コルーチンは非同期処理の様に振舞うが、非同期処理ではない。

戻り値はnullで良いが、WaitForSecondsクラスを使うことで遅延を発生させることもできる。Updateメソッド内で処理を行うと、毎フレーム処理が行われることになるが、コルーチン内でyield return WaitForSecondsクラスのインスタンス　とすれば毎フレームではなく、一定時間ごとに処理する仕組みを用意できる。すごく便利なのでぜひ覚えて欲しい。

コルーチン - Unity マニュアル
コンテキスト キーワード yield - C# リファレンス
UnityEngine.WaitForSeconds - Unity スクリプトリファレンス
MonoBehaviour-StartCoroutine - Unity スクリプトリファレンス

TIPS　StartCoroutineの引数

StartCoroutineメソッドは引数にIEnumerator型の値か文字列（メソッド名）を指定する。詳しくはリファレンス参照。

MonoBehaviour-StartCoroutine - Unity スクリプトリファレンス

連続投入の制限

（特記事項なし）

TIPS　コルーチン内のローカル変数

同じコルーチンを複数開始することができ、複数開始された時はそれぞれがローカル変数の値を保持する。

4-6　エフェクトの表示

パーティクルエンジンShuriken上でパーティクルを生成する方法を学ぶ。

パーティクルシステムの生成

パーティクルシステム

パーティクルは移動したり表示される画像やメッシュ。パーティクルシステムコンポーネントにて複数のパーティクルがまとまって制御される。俗に言うエフェクトが相当する。

パーティクルシステムは以下の手順で作成できる。

ヒエラルキーを右クリック　＞　エフェクト　＞　パーティクルシステム

パーティクルシステムとは何か？ - Unity マニュアル

パーティクルシステムの基本設定

ビルボード

パーティクルはどの角度から見ても見え方が変わらない。この仕組みを**ビルボード**と呼ぶ。

パーティクルシステムのモジュール

パーティクルシステムはいくつかの**モジュール**から構成されている。**メインモジュール**はパーティクルシステムの名称をクリックすると開閉する部分。他のモジュールはそれぞれ名称が付いたところをクリックすると開閉する。

図4-15　パーティクルシステムメインモジュール

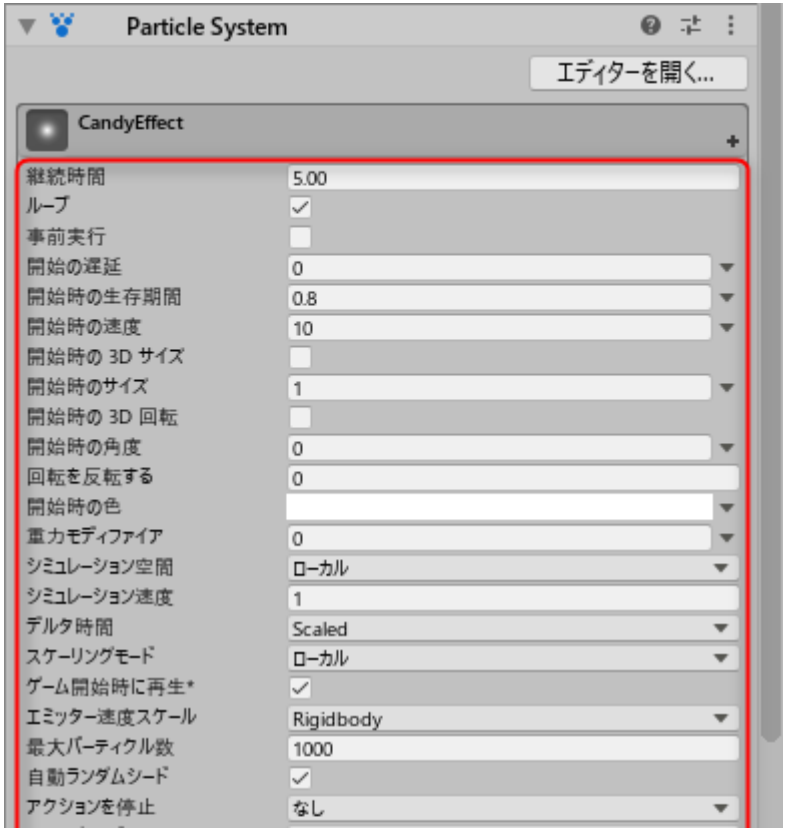

Particle System
エディターを開く...
CandyEffect
継続時間
5.00
ループ
事前実行
開始の遅延
0
開始時の生存期間
0.8
開始時の速度
10
開始時の 3D サイズ
開始時のサイズ
1
開始時の 3D 回転
開始時の角度
0
回転を反転する
0
開始時の色
重力モディファイア
0
シミュレーション空間
ローカル
シミュレーション速度
1
デルタ時間
Scaled
スケーリングモード
ローカル
ゲーム開始時に再生*
エミッター速度スケール
Rigidbody
最大パーティクル数
1000
自動ランダムシード
アクションを停止
なし

表4-1　パーティクルシステムのモジュール

メインモジュール	
開始時の生存期間（Start Lifetime）	出現してから消えるまでの時間
開始時の速度（Start Speed）	放出スピード
放出（Emission）モジュール	
時間ごとの率（Rate）	1秒間当たりのパーティクル放出量
形状（Shape）モジュール	
形状（Shape）	放出するパーティクルの形状
角度（Angle）	円錐の角度。形状がConeの場合に設定できる

半径（Radius）	形状の半径

Particle System メインモジュール - Unity マニュアル
Emission モジュール - Unity マニュアル
Shape モジュール - Unity マニュアル

COLUMN　パーティクルシステムの基本的プロパティ

非常に多くのプロパティがあるので、以下リンクから参照すること。

パーティクルシステムモジュール - Unity マニュアル

パーティクルマテリアルの設定

パーティクルのシェーダー

パーティクルにはStandardシェーダーではなく、Mobile/Particles/AdditiveやParticles/Standard Surface等の**専用のシェーダー**を使う。

パーティクルのスタンダードシェーダー - Unity マニュアル

パーティクルシステムのRendererモジュール

Rendererモジュールではマテリアルの設定を行うことができる。

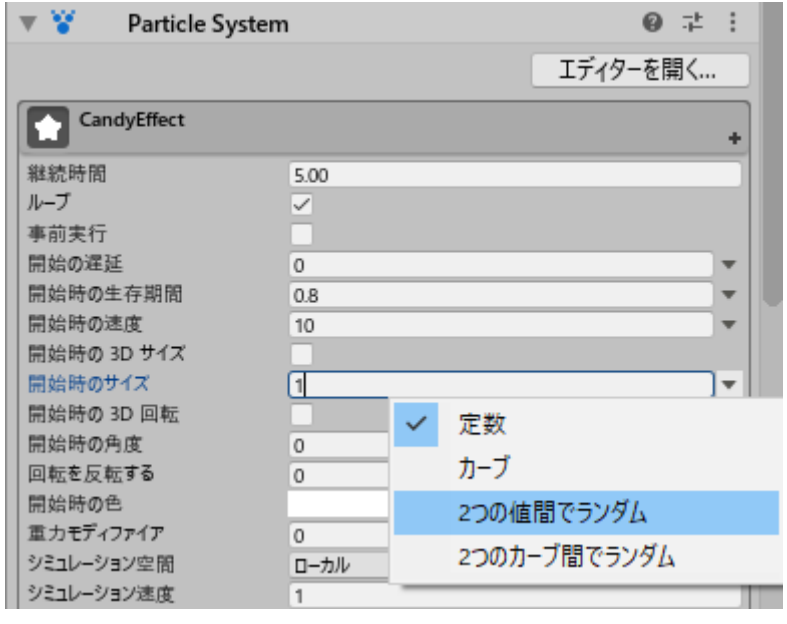
Renderer モジュール - Unity マニュアル

ランダム性の付与

ランダムな大きさのパーティクル

パーティクルの大きさをランダムにするには、パーティクルシステムメインモジュールの**開始時のサイズ（Start Size）**のオプションの**2つの値間でランダム（Random Between Two Constants）**を選択し、範囲を入力する。

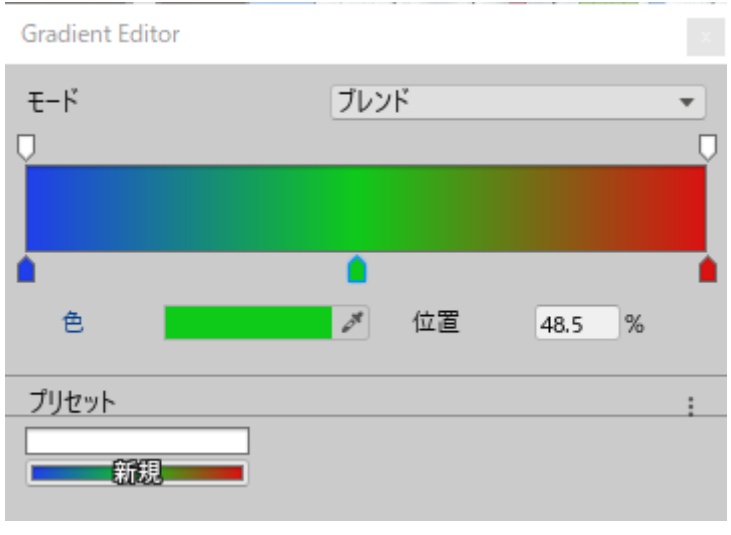
図4-16　開始時のサイズ



ランダムな色のパーティクル

パーティクルの色をランダムにするには、パーティクルシステムメインモジュールの開始時の色（Start Color）のオプションのランダムな色（Random Color）を選択し、グラデーションボックスを選択してグラデーションを編集する。

図4-17　グラデーションの編集



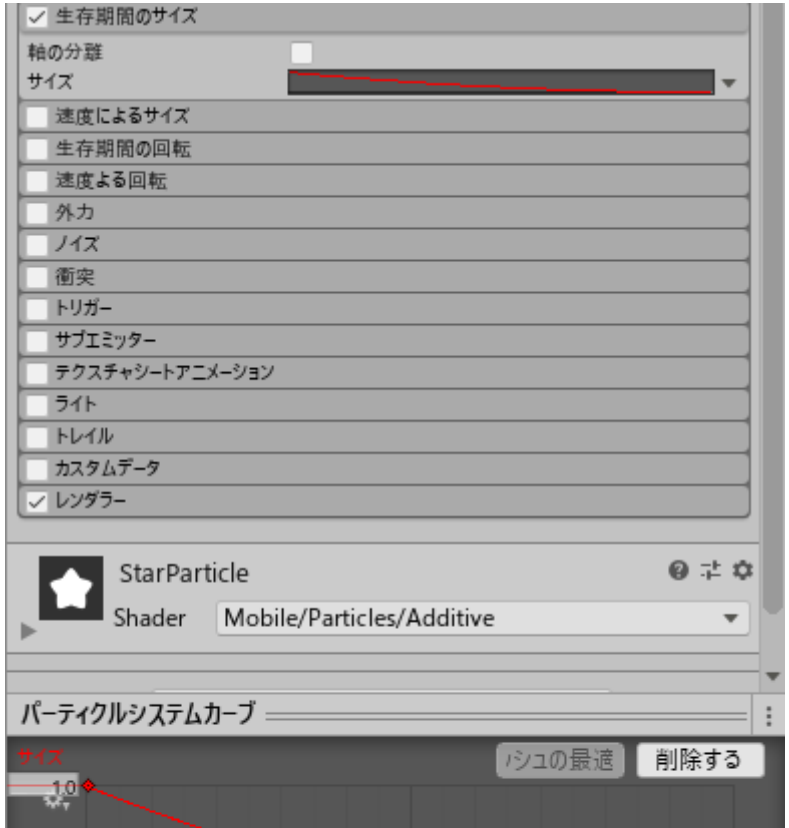
Lifetimeでの変化

生存期間によって大きさを変化させる

生存期間によって大きさを変化させるには、パーティクルシステム**生存期間のサイズ（Size over Lifetime）**モジュールにチェックを入れ、サイズプロパティの右にあるカーブ設定領域をクリックし、パーティクルシステムカーブを編集もしくはプリセットから選ぶ。

Size Over Lifetime モジュール - Unity マニュアル

図4-18　生存期間のサイズモジュール


生存期間のサイズ
軸の分離
サイズ
速度によるサイズ
生存期間の回転
速度よる回転
外力
ノイズ
衝突
トリガー
サブエミッター
テクスチャシートアニメーション
ライト
トレイル
カスタムデータ
レンダラー
StarParticle
Shader
Mobile/Particles/Additive
パーティクルシステムカーブ
サイズ
1.0
削除する

生存期間によって回転を変化させる

生存期間によって回転を変化させるには、パーティクルシステム**生存期間の回転（Rotation over Lifetime）**モジュールにチェックを入れ、角速度を入力する。

Rotation Over Lifetime モジュール - Unity マニュアル

（復習再開ポイント）

エフェクトの再生秒数と自動削除の設定

パーティクルシステムのループ設定

パーティクルシステムメインモジュールの**ループ（Looping）**のオン・オフでループするかどうかを設定できる。

パーティクルシステムの再生時間設定

パーティクルシステムメインモジュールの**継続時間（Duration）**で再生時間を設定できる。

パーティクルシステムの再生終了後設定

パーティクルシステムメインモジュールの**アクションを停止（Stop Action）**で**破棄**を選ぶと、再生終了時にパーティクルのオブジェクトを自動的に削除するようになる。

パーティクルシステムをプレハブ化

パーティクルシステムはプレハブ化することができる。

TIPS　パーティクルのプレビュー

パーティクルシステムを選択後、シーン右下に**パーティクルエフェクト**ウィンドウが表示され、パーティクルをプレビュー操作することができる。

図4-19　パーティクルエフェクト


シーン
アセットストア
Shaded
2D
ギズモ
All
Persp
CANDY BOX
CANDY BOX
パーティクルエフェクト
一時停止
再起動
停止

エフェクトの生成

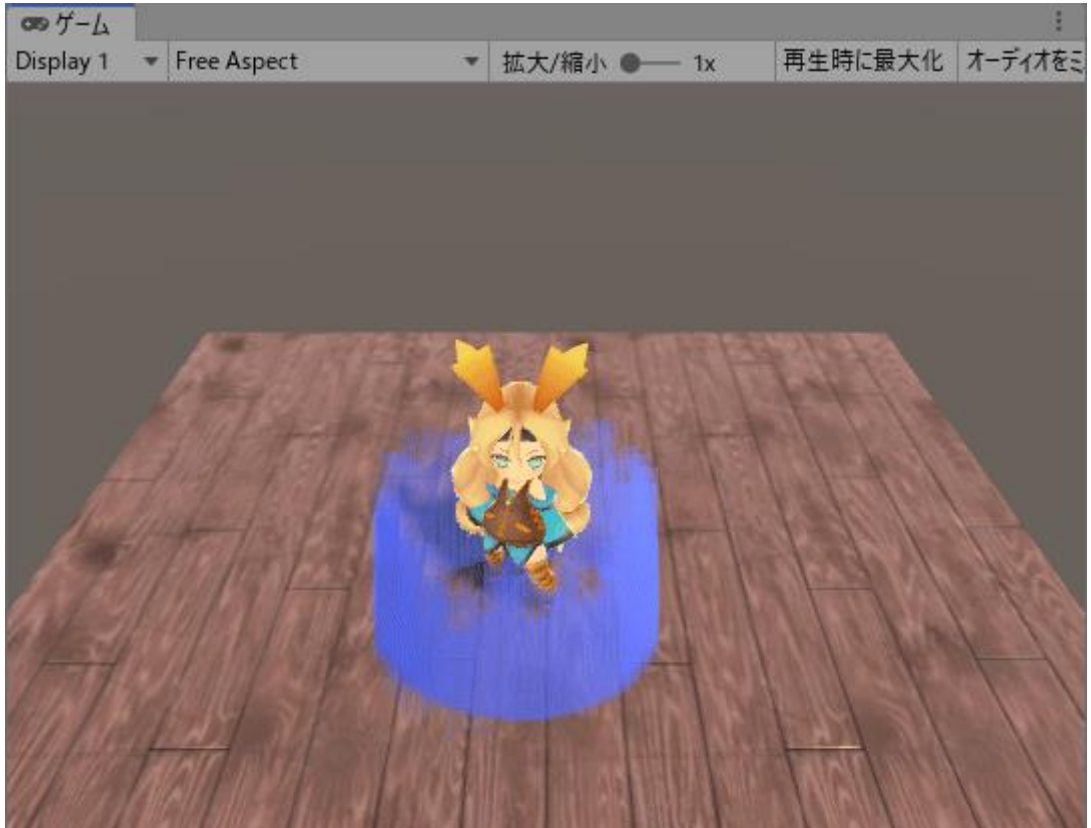
Instantiateメソッド（回転あり）

回転した状態で動的にオブジェクトを生み出すにはQuaternion.identityではなく、Quaternion.Eulerメソッドを使う。X、Y、Z軸の任意の角度を代入したVector3構造体を引数に指定する。例えば、X軸に180°回転する場合はQuaternion.Euler(new Vector3(180, 0, 0))となる。

Quaternion-Euler - Unity スクリプトリファレンス

おまけ

パーティクルはアルベドのアルファ値を指定した（半透明な）色を使うと、テクスチャなしでもそれなりの見た目を作ることができる。多少のランダム性、生存時間による変化を付けるとより良い。


ゲーム
Display 1
Free Aspect
拡大/縮小
1x
再生時に最大化
オーディオをミ

4-7　サウンドの再生

Unityのオーディオコンポーネント

`Audio Source`

Audio Sourceは音源のコンポーネント。

オーディオソース - Unity マニュアル

`Audio Listener`

Audio Listenerはサウンドを聞くためのコンポーネント。メインカメラに最初からアタッチされている。

Audio Listener - Unity マニュアル

COLUMN　シーンの音響を再現する3Dサウンド

UnityはAudio Sourceが発する音がどの方向にあるのか再現する。向かって左に音源がある場合は左スピーカーから、右側にある場合は右スピーカーから聞こえるようになる。音源が遠い場合は音が小さくなる。

エフェクト音の再生

サウンドを再生（スクリプト制御なしで再生）

サウンドを再生するにはオブジェクトにAudio Sourceコンポーネントをアタッチし、**オーディオクリップ**プロパティに音源を指定する。

プロパティの**ゲーム開始時に再生**にチェックをつけると、そのオブジェクトがシーンに追加された時に自動でサウンドが再生される。

スクリプトからの任意タイミングでの再生

サウンドを再生（スクリプト制御ありで再生）

スクリプト上でサウンドを再生するには、AudioSourceコンポーネントを取得し、**Playメソッド**でサウンドを再生できる。

AudioSource-Play - Unity スクリプトリファレンス

※リファレンスには引数ありのPlayメソッドしか存在しないが、実際には引数なしのPlayメソッドが存在する。

BGMの再生

ループ再生

Audio Sourceコンポーネントのプロパティのループにチェックをつけると、ループ再生することができる。

COLUMN　サウンドのインポート設定

プロジェクトビューからサウンドを選択すると、インポート設定がインスペクタ上で表示される。サウンドは容量が大きくなりがちなので、用途・サウンド特性に合わせて適切な設定をすること。

表4-2　サウンドのインポート設定

ロードタイプ	
ロード中に解凍	オーバーヘッドが少ない。頻繁に使われるサウンド用。
圧縮状態でメモリに格納	サウンド再生時に解凍する。オーバーヘッドが若干発生するので、稀に使われるサウンド用。
ストリーミング	ストリーミング再生する。BGM等の長いサウンド用。
圧縮形式	

PCM	サウンドに圧縮をかけない。効果音用。
Vorbis	Vorbis形式で圧縮する。若干の劣化が発生する。
ADPCM	ADPCM形式で圧縮する。Vorbis形式より圧縮率は低い（=サイズが大きくなる）。劣化が少ない。

[オーディオファイル - Unity マニュアル]

Audio Mixerの利用

オーディオミキサー

以下の手順でオーディオミキサーを利用することができる。

ウィンドウ　＞　オーディオ　＞　オーディオミキサー

オーディオミキサーにはグループの概念があり、Masterが最初からグループに存在している。Masterをマスター音源として子グループを作り、それぞれをフェーダー（上下に動かせるツマミのこと）で出力レベルを調整することでグループ別音量調整が可能。

Audio Mixer - Unity マニュアル

まとめ　アセット管理とゲームオブジェクトの制御

- アセットはインポートしたりエクスポートしたりできる。
- マテリアルでテクスチャを扱うことができる。
- プレハブを使うことでアセットをひとまとめにすることができる。
- Instantiateメソッドで動的にオブジェクトを生み出すことができる。
- コルーチンを使えば一時的に処理を中断して次フレームで再開することができる。
- コルーチンは再開を次フレームではなく、秒数遅延後に再開することもできる。
- コルーチンは複数開始することができ、それぞれがローカル変数を保持する。
- パーティクルシステムを使うことで、パーティクルを生成できる。
- パーティクルシステムにはいくつかのモジュールがあり、それぞれ役割を持つ。
- Audio Sourceコンポーネントを使ってサウンドを再生することができる。
- オーディオミキサーを使って出力レベルを調整することができる。


UNITY★CHAN!
LICENSED UNDER UNITY-CHAN LICENSE