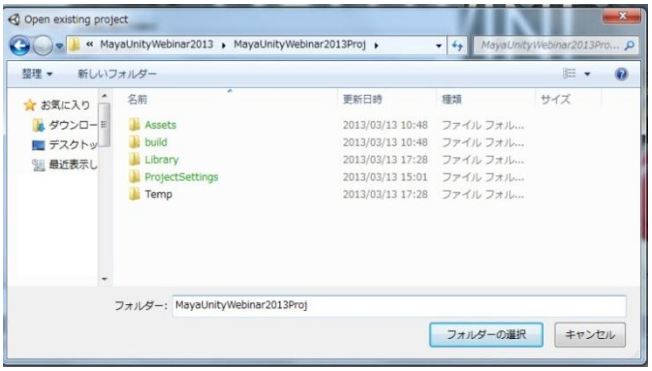
Autodesk Maya Unity Webinar 2013 プロジェクト説明

内容

プロジェクトの開き方



MayaUnityWebinar2013Proj フォルダが Unity のプロジェクトです。File > Open Project...でこのフォルダを開いてプロジェクトをセットして下さい。全てのスクリプトはプロジェクト内の Assets > Game > Scripts に入っています。説明の中で C#(.cs)ファイルの名前が出てきます。それらは Scripts フォルダに入っていますので、必要に応じて参照して下さい。

ゲームシーンを開く

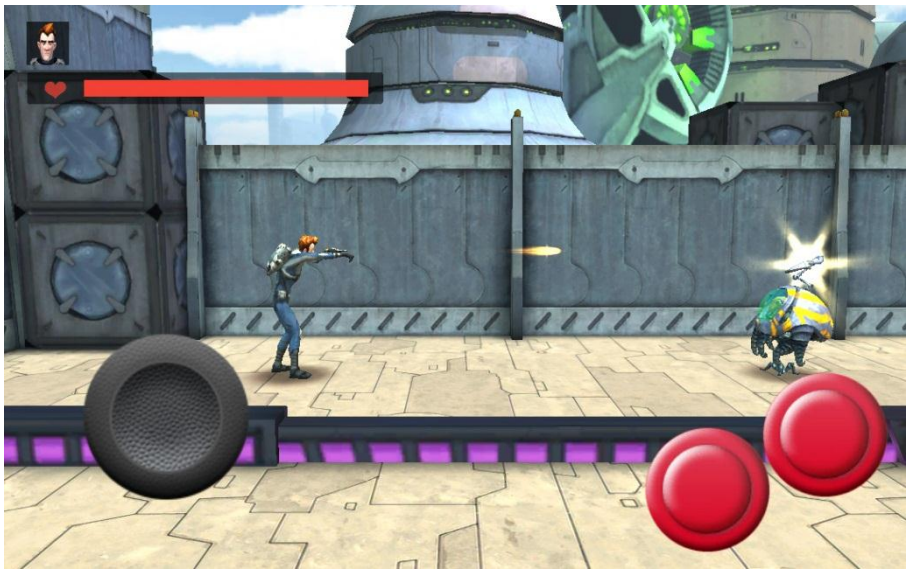
プロジェクトの設定が出来たら次にゲームのシーンを開きましょう。Assets > Game > Scenes > Level01.unity というシーンファイルを開いて下さい。本ゲームはこのシーンでできています。

シーン内の箱や壁を自由に配置してステージを好みに変えてゲームを楽しんでみて下さい。他のプロジェクトで作られたアセットをインポートしてシーンに配置すればゲームに組み込めますし、自分で作ったオブジェクトに置き換えれば見た目の全く違う、別のゲームにすることもできます。アセットを入れ替えて、パラメーターを調整して、ゲームの反応を見て楽しんでいるうちに自然と Unity の使い方にも慣れていくことでしょう。

このドキュメントではカメラやプレイヤーと言ったゲームで不可欠な各要素の概要を説明しています。読み進めればシーンがどんな風に作られているかわかってくることでしょう。慣れてきたらコードを開いて見ましょう。プロジェクトに含まれているコードは全て C#で用意されています。複雑な処理をさせ

るよりは、シンプルにゲームの流れを把握出来るように書かれていますので、編集も簡単に行えます。コードを変更してゲーム内での反応を見つつ、独自のゲームに進化させてみて下さい。

プレイの仕方



プレイヤーキャラを操作して敵を倒したり敵から逃げたりしながら、最終エリアにいるボスを倒すことが本ゲームのゴールです。ボスを倒すと画面に「YOU WIN!」と表示されます。敵の体当たりを食らうと左上のライフゲージが減っていきます。ライフがなくなるとゲームオーバーとなり、スタート地点に戻されます。

プレイヤーの操作は、キーボード入力、ゲームパッド入力、タッチ入力（モバイルもしくはマウスクリック）に対応しています。

キーボード入力
移動：WASD キーおよび矢印キー
ショット：左 Ctrl キー
ジャンプ：スペースキー

ゲームパッド入力
移動：十字キーもしくはアナログスティック
ショット：ボタン１
ジャンプ：ボタン２
（どのボタンをショット・ジャンプにするかはシーン内の Player オブジェクトの Locomotion Ctrl コンポーネントで定義しています）

タッチ入力
移動：バーチャルスティック
ショット：右赤ボタン
ジャンプ：左赤ボタン
（各 GUI はシーン内の GUICamera > virtualStick1, 2, 3 の Virtual Stick Ctrl コンポーネントで制御しています）

カメラの設定

ゲーム制作にはカメラ制御が不可欠です。良いカメラならユーザーはゲームに没頭できますが、カメラが変な動きをしているとゲームに集中できずにイライラしてしまいます。

ゲームのタイプによって適切なカメラというものがあります。本ゲームでは横スクロールから始まり、次に奥方向への移動もできるレベル構成（ステージ構成）になっていますので、以下の様な特徴を持つカメラにしてあります。

- プレイヤーの少し前方を見せる
- プレイヤーの高さに応じてカメラの高さも変える
- プレイヤーとカメラの間に余計なオブジェクトがある場合、カメラがめり込まないようにする

横スクロールアクションを作るなら今のカメラ設定のままで十分です。好みに応じて若干オプションを調整してみて下さい。キャラクターの後ろにカメラが付くサードパーソンビューやカメラ自身がキャラの代わりになるファーストパーソンビューにも対応出来ます。以下のようにパラメーターを設定してみて下さい。

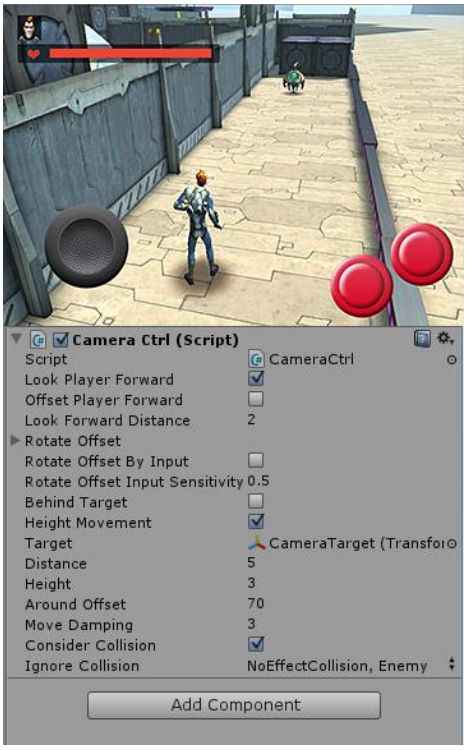
シーン内の Main Camera の Camera Ctrl コンポーネントの値を以下のように変えます。

横スクロールカメラ（デフォルト）

Look Player Forward = ON
Offset Player Forward = ON
Look Forward Distance = 2
Rotate Offset = 0, 0, 0
Rotate Offset By Input = OFF
Rotate Offset Input Sensitivity = 0.5
Behind Target = OFF
Height Movement = ON
Target = Player>Sven>CameraTarget
Distance = 8
Height = 1
Around Offset = 0
Move Damping = 3
Consider Collision = ON

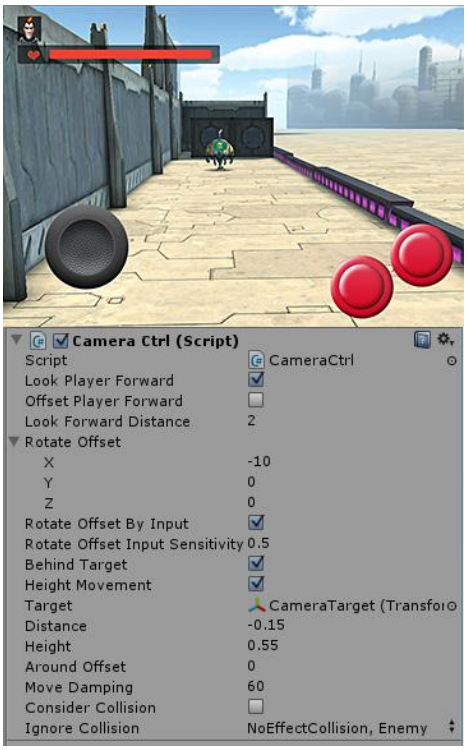
TPS 風カメラ

Look Player Forward = ON
Offset Player Forward = OFF
Look Forward Distance = 2
Rotate Offset = 0, 0, 0
Rotate Offset By Input = OFF
Rotate Offset Input Sensitivity = 0.5
Behind Target = OFF
Height Movement = ON
Target = Player>Sven>CameraTarget
Distance = 5
Height = 3
Around Offset = 70
Move Damping = 3
Consider Collision = ON



FPS 風カメラ

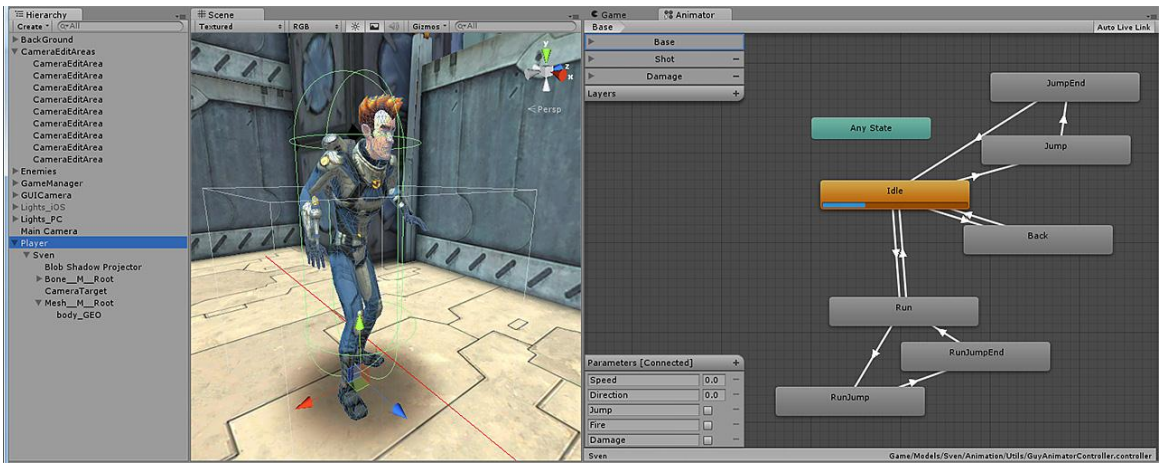
Look Player Forward = ON
Offset Player Forward = OFF
Look Forward Distance = 2
Rotate Offset = -10, 0, 0
Rotate Offset By Input = ON
Rotate Offset Input Sensitivity = 0.5
Behind Target = ON
Height Movement = ON
Target = Player>Sven>CameraTarget
Distance = -0.15
Height = 0.55
Around Offset = 0
Move Damping = 60
Consider Collision = OFF



FPS 視点の場合はプレイヤーモデルがカメラに写り込まないようにするために Player > Sven > Mesh__M__Root > body_GEO の Skinned Mesh Renderer のチェックを外します。プレイヤーの動きがカメラの動きに連動しますので Player の Locomotion Ctrl の Direction Is Chara Space をオンにします。これで入力した方向とカメラ、プレイヤーの方向が一致するようになります。

詳しいカメラの処理は CameraCtrl.cs を参照して下さい。このスクリプトをベースに作りたいゲームに合ったカメラになるよう余分なコードを削除したり改造してみて下さい。別のプロジェクトにカメラを移行するのなら、CameraCtrl.cs をコピーして持って行き、シーン内のカメラに追加するだけでできます。

プレイヤーの動き



シーン内の Player オブジェクトがプレイヤーです。ユーザーの入力に対する反応は Locomotion Ctrl コンポーネントで行なっています。アニメーションと衝突判定は Player > Sven オブジェクトの Animator と Character Controler で行われています。

Unity ではプレイヤーを動かすのに様々なアプローチがあります。従来のアニメーションシステムを使用することもできますし、Unity4 から搭載された Mecanim でアニメーションさせることもできます。本ゲームでは Mecanim を使用しています。

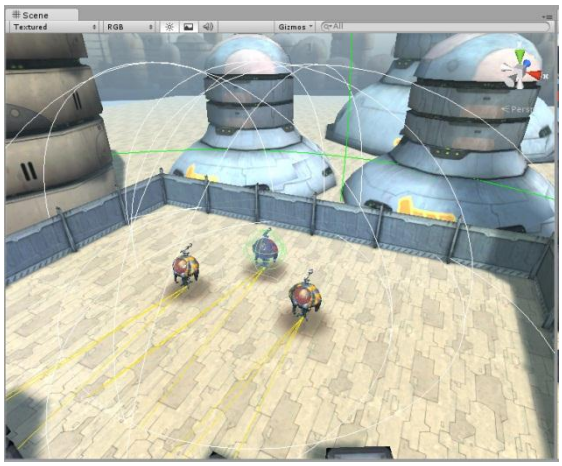
従来通りの方法でプレイヤーを移動させる場合、ユーザーの入力に応じてプログラムで Transform を更新して移動します。一方 Mecanim を使用する場合は、アニメーションの中でキャラクターが移動していれば、自動的に Transform に値が設定されます。本ゲームでのプレイヤーは移動とジャンプを Mecanim に任せているので、移動速度、ジャンプ距離・高さを変更するには元の FBX ファイルのモーションを変更する必要があります。

プレイヤーの回転の制御は Mecanim での処理と、従来通りのプログラムによる方向制御の両方を実装しています。それぞれで入力に対する反応の感触が異なりますので、作りたいゲームに合わせて選んでみて下さい。Player オブジェクトの Locomotion Ctrl コンポーネントの Rotate By Mecanim のチェックを変えて変更できます。Mecanim を使用するとリアリティのあるモーション補間がされます。従来通りの方法なら、入力に対して瞬間的に反応させることができます。

ユーザーの入力はワールド座標の方向です。AD キー（もしくは左右）なら X 軸方向に、WS キー（もしくは上下）なら Z 軸方向に動きます。Locomotion Ctrl コンポーネントの Direction Is Chara Space をオンにすると入力はキャラの方向になります。左右なら左右に、上下なら前後に動くようになります。FTP ゲームでは通常この操作でプレイヤーを動かします。

詳しくは LocomotionCtrl.cs を参照して下さい。

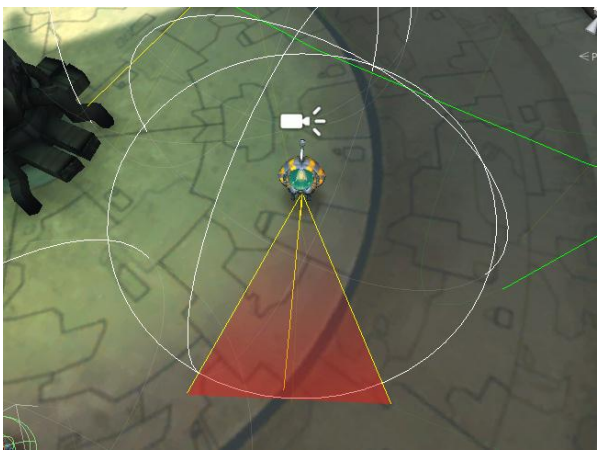
敵の動き



本ゲームでは minion という敵キャラがいます（プレハブはプロジェクト内の Assets > Game > Models > minion_wip > Prefabs > minion です）。このモデルの色や大きさを変えて一般兵、ちょっと強いやつ、ボス、という三種類の敵を用意しています。どれも Enemy Locomotion Ctrl コンポーネントを使っており、パラメーターを調整することで攻撃にバリエーションをつけています。

敵ごとにバラバラの AI を作るのではなくパラメーター調整で違いを出すといろいろな恩恵を得られます。プログラムが楽になったり、バグも少なくなるだけでなく、敵の動きのパターンに一貫性が出て、ユーザーにとっても理解しやすい、徐々に慣れて攻略しやすいものになります。敵ごとに動きが異なるとパターンを読み取ることが難しくなり、とても難しいゲームになります。同じような動きをするけれど、強い敵は動きが機敏になる、もしくはある部分に隙ができて倒しやすくなるという調整をする方が遊びやすくなり、またゲームが進んでいる感も出てきます。

Enemy Locomotion Ctrl コンポーネントの値を変更して AI の強さを制御できます。本ゲーム用での AI はとてもシンプルです。敵は常に自分の中心から Player Search Area 分離れた範囲にプレイヤーがいないかチェックしています。このサーチエリア内にプレイヤーが入ってきたら Look Player Speed のスピードでプレイヤーの方向を向きます。Angle To Attack で指定した範囲内にプレイヤーを捉えたら、体当たり攻撃を始めます。プレイヤーが敵の正面から逃げ出せば敵はプレイヤーを見失い、攻撃を止めます。



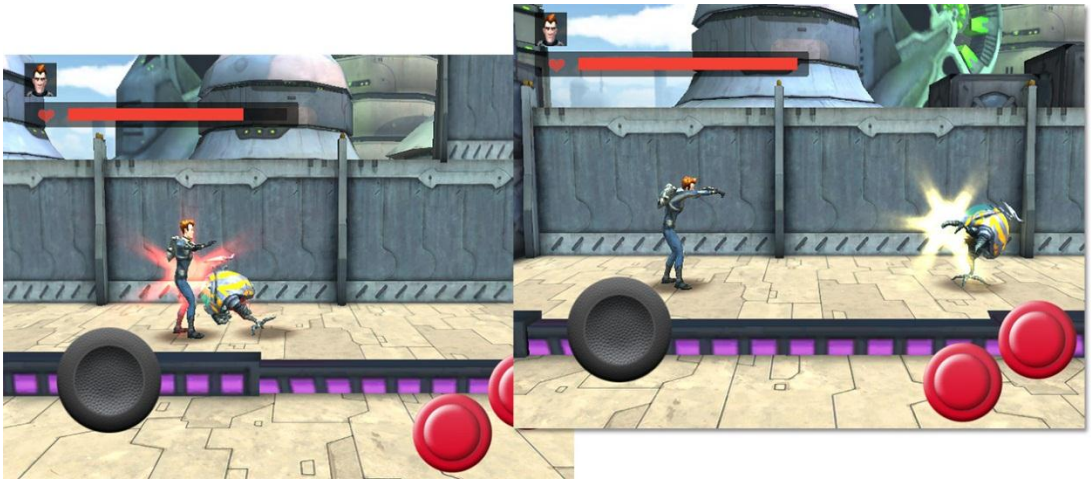
敵はダメージを食らうと Damage Wait の分だけ放心します。その隙にプレイヤーは攻撃したり逃げ出したりできます。

Random Walk をオンにすれば敵は適当に歩きまわります。もし止まっていれば、歩く方向と時間を決めて歩き始めます。また止まれば、また歩く方向と時間を

ランダムに決めます。移動アニメーションは Mecanim で行われます。回転の制御は Enemy Locomotion が回転値を Transform に直に設定しています。回転の速度が難易度に関わってくるので、モーションで修正するよりはパラメーターで簡単に制御できる方が調整しやすいからです。

詳しくは EnemyLocomotionCtrl.cs を参照して下さい。

ダメージの処理



ダメージの与え方は攻撃方法やプレイヤー、エネミーによって異なるように思えますが、基本的なやり取りは同じでシンプルです。「HP を保持するコンポーネント」「HP を削るコンポーネント」という二つのやり取りで成り立っています。HP が減った時に HP が 0 になったかを確認し、ダメージモーションを再生するか、死亡とするか決めます。ダメージを与えるものが弾だったり体当たりだったりボムだったりするので、ダメージを与える側にはいくつかのパターンがあります。

HP を保持するコンポーネントとして、本ゲームでは HPCtrlBase.cs というスクリプトを用意しています。HP 管理の基本となるスクリプトです。このスクリプトの派生として EnemyHPCtrl.cs と PlayerHPCtrl.cs があります。それぞれ敵とプレイヤーの HP コントロールをします。敵は単にダメージを受けるだけで良いのですが、プレイヤーはダメージを受けた後一定時間無敵になったり、ライフゲージと連携させる必要があります。

HP を削るコンポーネントには BulletCtrl.cs と EnemyBodyAttackCtrl.cs があります。BulletCtrl.cs はプレイヤーが撃つ弾の制御を行います。プロジェクト内の Assets > Game > Effects > Prefabs > Bullet プレハブでこのコンポーネントを使用しています。Player オブジェクトの Locomotion Ctrl コンポーネントがユーザーのボタン入力を確認し、弾を発射することになったら Shot Ctrl コンポーネントを実行します。すると Bullet プレハブがシーン内にインスタンスされ、指定された速度で飛び出していきます。Target Layer に指定されたレイヤーのオブジェクトのコライダーと衝突すると、衝突相手の HP コンポ

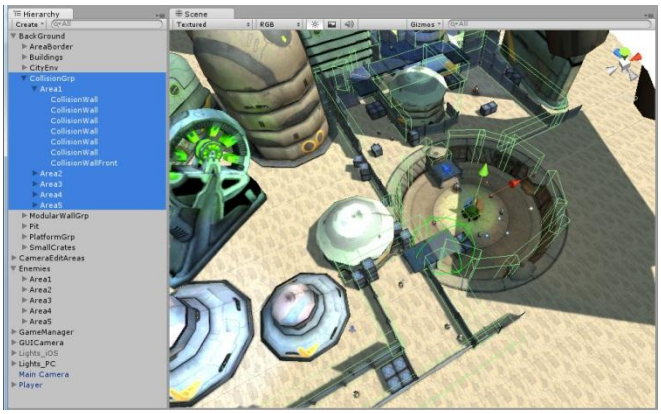
ーネントを見つけて、ダメージ分 HP を減らします。

EnemyBodyAttackCtrl.cs は敵に設定してあり、ぶつかった相手がプレイヤーなら HP コンポーネントを見つけてダメージを与えます。

おまけで EasterEggBomb.cs というスクリプトが用意してあります。キーボードで「unitea」と入力すると発動し、プレイヤーの距離から一定距離内の敵を見つけ出し、HP コンポーネントを取得して大ダメージを与えます。

このように攻撃方法が違っても、攻撃対象を見つけて、そのオブジェクトの HP コンポーネントを見つけてダメージをセットするというのは、どのゲームでも使える基本的なテクニックです。

衝突判定



いわゆる「あたり判定」「コリジョン判定」といわれるものです。Unity ではオブジェクトにコライダーを設定すれば、自動で衝突を検知してくれます。各オブジェクト、例えば壁の一枚一枚にコライダーを用意することもできますが、今回は大きな Box Collider をステージ内の各場所に配置しています。処理負荷を減らすとともに、コリジョン抜けといわれる、キャラクターがコライダーの隙間を抜けてしまうことを避けるためにもこのシーンでは効果的です。

BackGround > CollisionGrp 内にシーンで使っているコライダーが含まれています。もちろん場合によっては個別のオブジェクトにもコライダーが含まれています。シーンを編集するのに壁をコピーしてもコライダーが含まれていませんので、必要に応じてコライダーを追加してください。

コライダーの大きさは対象物よりちょっと大きめにしておくのがポイントです。特に敵のコライダーが敵のモデルのサイズと同一だと、思った以上に弾が当たりません。ちょっと大きめにしておいたほうがゲームしやすくなります。

ゲーム全体の処理

これまで説明してきたゲームの要素は、各オブジェクトで独立して動いているものです。カメラはカメラだけで完結して勝手に動いていますし、プレイヤーも敵も独立して勝手に動いています。

しかしゲームではシーン全体をチェックしてゲームの進行を管理するまとめ役が必要です。シーン内の GameManager オブジェクトに付いている Game Manager コンポーネントがその役割を担っています。

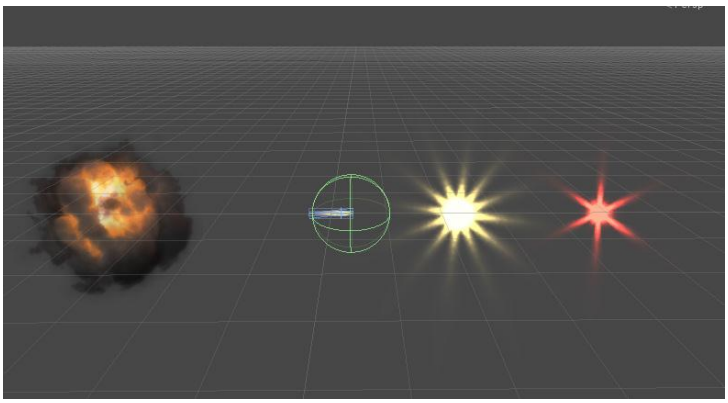
Game Manager コンポーネントは、ライフゲージとボスのチェックを行なっています。ライフゲージがなくなれば、コンポーネントに登録してある「GAME OVER」の文字を表示し、ボスが死んでいれば「YOU WIN!」の文字を表示してゲームを再スタートします。

更にこのオブジェクトではシーン全体に関わる機能を追加しています。Runtime Settings コンポーネントは、ゲームスタート時に Ambient Light を設定します。ライティングをベイクする時とゲーム実行時で Ambient Light の明るさを変えたいのでこの処理を加えています。もし他にもゲームスタート時に設定したいことがあれば、このスクリプトを拡張すると便利です。

Easter Egg Bomb コンポーネントはいわゆる「バグ技」です。「unitea」とキーボードで入力すると、プレイヤーから Range 範囲内の敵に 10000 のダメージを与えます。つまり、敵を一掃します。

必要に応じてシーン全体に関わるコンポーネントをこのオブジェクトをまとめておくと便利です。

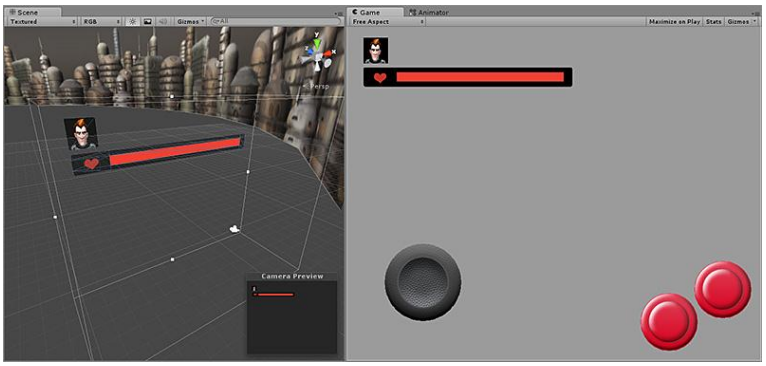
エフェクト処理



ゲーム実行中に様々なコンポーネントが、様々なタイミングでエフェクトをインスタンスします。シーン内にどんなエフェクトがあって、どれを削除するべきかなど管理するのは大変です。大抵のエフェクトは一定の時間表示したら不要になります。不要かどうかは、各エフェクトが管理するようにすれば漏れ無くエフェクトを管理出来ます。

全てのエフェクトはプロジェクト内の Assets > Game > Effects > Prefabs 内に入っています。Bullet 以外のエフェクトには全て Self Destroy コンポーネントが設定されています。Lifespan で指定した秒数が経つと自身を削除します。Bullet は BulletCtrl.cs 内で自滅するようにしてあります。これなら確実に無駄なエフェクトが残らないようにできます。

GUI



本ゲームは簡単なアクションゲームですので凝った GUI は必要ありません。シーン内の GUICamera オブジェクトが GUI 表示用のカメラです。全ての GUI 関係のオブジェクトはこのカメラの子供にして管理しています。

カメラを GUI 用にするにはカメラのパラメーターを調整します。Clear Flags を Depth only に、Depth を 1 にして、通常のビューの後に追加で描画されるようにします。Culling Mask を設定して、必要な GUI 素材だけが描画されるようにします。Projection を Orthographic にしてパースがかからないようにし、GUI が歪まないようにします。より複雑な GUI を用意する時もこういった専用カメラを利用すると便利です。

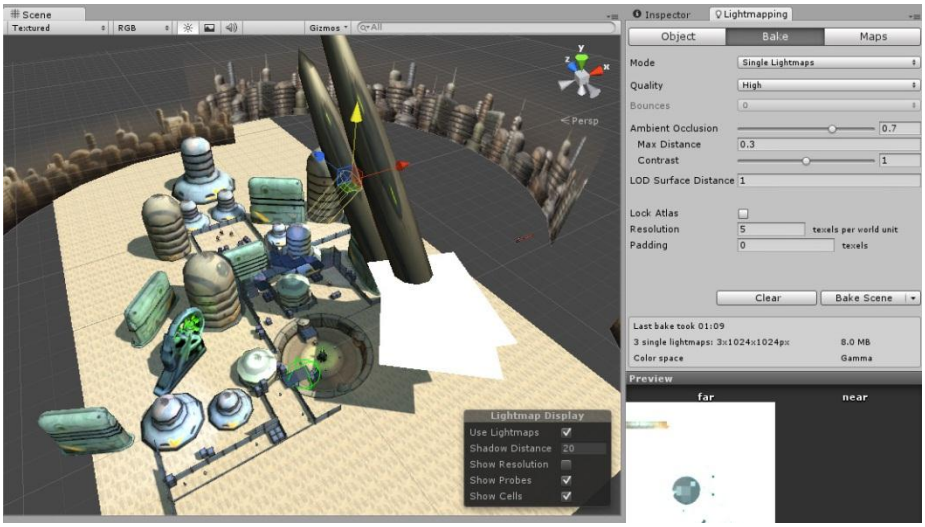
カメラの自動切り替え

市販のゲームで遊んでいるとステージ進行に応じて遊びやすいようにカメラの位置が変わります。本ゲームでもゲームを進めていくと、所々でカメラの高さが変わります。シーン内の CameraEditAreas オブジェクトの子供のオブジェクトでこの制御を行なっています。

各オブジェクトには Camera Edit Area Status コンポーネントがついています。同オブジェクトに付いている Box Collider 内にプレイヤーが入ってきたら、メインカメラの Camera Ctrl コンポーネントにアクセスして値を変えます。HP の操作に似ています。

カメラがエリア内に来たら何かを実行する、というスクリプトですのでこれを調整することで
・カメラのアニメーションを再生してカットシーン演出を流す。
・別のレベルに移動する。
など様々なことに応用出来ます。

ライティング



本ゲームでは Unity 無料版を想定して制作されています。シーン内に Lights_PC、Light_iOS という名前のオブジェクトがあり、使用している全てのライトが含まれています。それぞれ PC 用と iOS 用のラ

イトです。

Lights_〇〇 > EnvLights にはシーン全体に影響を与えるライトが入っています。KeyLight を太陽光としています。これが基本的な明るさ、光の方向、影の方向を決め、シーンの印象を決定づけます。これをベースに空をシミュレートする SkyReflectionLight という Directional Light が複数あります。地面の照り返しは GroundReflectionLight で行なっています。KeyLight はリアルタイム描画にも使用します。ベイクの時だけ明るさを更に調整するために KeyLight_bake というライトを用意しています。ライトマップのベイクには Edit > RenderSettings の Ambient Light の明るさも影響しますので注意して下さい。

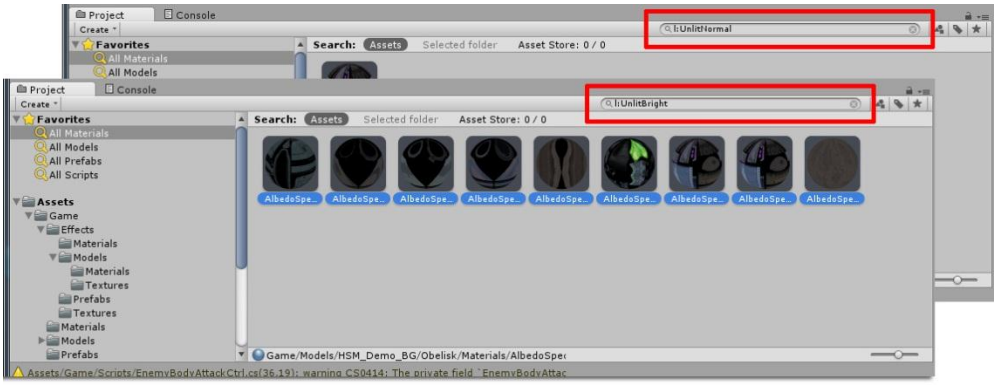
Lights_〇〇 > Area〇にはエリアごとの独自のライティングが設定されています。全体のライティングの傾向を決めた上で、細かいライトの調整を行なっています。

Lights_〇〇 > EnvLights > HideWhenRun 内にあるオブジェクトはライトマップに影を落とすためのオブジェクトで、ゲーム実行時には Culling Ctrl コンポーネントによって非表示にされます。KeyLight で大まかなライティングの方向性を決めた上で、このように影の落ち方を調整します。ライティングの美しさは明るい所ではなく、むしろ影に現れますのでこのように影用オブジェクトを追加して、ステージが魅力的になるよう調整していきます。影が沢山落ちていると、実際にはシーン内にそれほどオブジェクトがなくても沢山あるように感じますので、モバイル向けのゲームでシーンの密度を演出するのにも都合が良いです。

今回のシーンでは iOS でベイクしたライトマップの明るさや傾向が PC と異なるので、Lights_PC と Lights_iOS の二つで別管理にしてあります。Lights_PC と比べて Light_iOS 内のライトは全て明るさが半分以下になっています。ライトの数などは同じです。iOS ではカスタムのシェーダーを使用し、ライトマップを明るく描画するようにしています。PC と iOS で以下の対応が必要です。

PC 環境

- Lights_PC をアクティブに、Lights_iOS を非アクティブにします。
- ライトマップをベイクし直します。
- Project ウインドウで「l:UnlitBright」と入力してマテリアルを検索します。
- 見つかったマテリアルのシェーダーを Mobile/Unlit Bright(Supports Lightmap)から Bumped Specular に変更します。
- Project ウインドウで「l:UnlitNormal」と入力してマテリアルを検索します。
- 見つかったマテリアルのシェーダーを Mobile/Unlit(Supports Lightmap)から Bumped Specular に変更します。



iOS 環境

・　Lights_PC を非アクティブに、Lights_iOS をアクティブにします。

・　ライトマップをベイクし直します。

・　Project ウインドウで「l:UnlitBright」と入力してマテリアルを検索します。

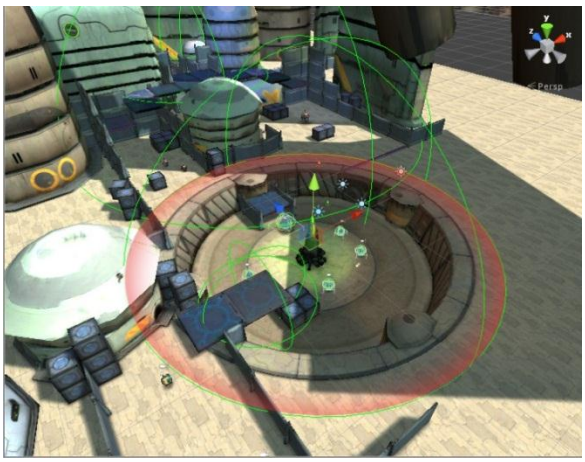
・　見つかったマテリアルのシェーダーを Bumped Specular から Mobile/Unlit Bright(Supports Lightmap)に変更します。

・　Project ウインドウで「l:UnlitNormal」と入力してマテリアルを検索します。

・　見つかったマテリアルのシェーダーを Bumped Specular から Mobile/Unlit(Supports Lightmap)に変更します。

パフォーマンス

沢山のオブジェクトを出して、リッチなシェーダーを使って、大きなテクスチャを用意して、なるだけ見た目をキレイにしたいところですが無限に描画できるわけではありません。特にモバイル向けのゲームではパフォーマンスが処理落ちに影響をあたえるだけでなく、消費電力にも影響を与えますのでバッテリーの持ちや本体の発熱にも関わってきます。



処理のパフォーマンスを良くするには、少ないオブジェクト、簡易なシェーダー、小さなテクスチャを使うことが重要です。本ゲームではパフォーマンス調整のひとつのサンプルとしてカリング処理を提供しています。

シーン内の敵モデルをまとめている Enemies > Area2, 3, 4, 5 に Culling Ctrl コンポーネントがついています。このコンポーネントはカメラかプレイヤーの距離が

Culling Distance よりも外である場合は子のオブジェクトを非表示にします。プレイヤーが進んでいくごとに必要な分だけ敵を表示することで、処理負荷を軽減しています。

ビルド

File > Build Settings...でターゲットになるプラットフォームを選択してビルドします。PC や Web 用ならそのままビルドします。iOS 向けの場合は「ライティング」項目に書かれている iOS 環境用のライティング、シェーダー設定を行った後にビルドします。

iPhone などに出力する場合は XCode のインストール、アップルのサイトでのデベロッパー登録など必要になりますので、Unity のドキュメントを参考にして下さい。