










ゲームの UI を構築する

UI（ユーザー・インターフェイス）はゲームで最も重要な要素のひとつです。典型的なゲームでは、ライフや、有り金の量といったプレイヤーの状態を示す情報が画面上の決められた位置（HUD）に表示され、そういったUIから情報を拾い上げてゲームを攻略していくことは、ゲームプレイの大きな楽しみのひとつです。

2015年現在では、本来UIの絵素材を作ることを生業とするアーティストが、より良いゲーム体験のためにあえてUIを減らす決断をする光景も珍しいものではなくなりつつあります。もはや現在のUI従事者は、たくさんの量のUI絵素材を請け負う絵描きではありません。盆栽を嗜む人が、あえて枝葉を切り落とすことで本質をえぐり出すように、必要なものを必要な形でゲームに持ち込むことを検討する裁量権と能力を持つUIの**設計者（デザイナー）**として、その手腕を日々発揮しています。その職責は、ゲーム全体の情報の置き方や、体験の手順の設計にまで及ぶことがあり、この分野は近年、**UX**として知られる、押しも押されぬ専門職のひとつになっています。

本章では、UI開発のワークフローに簡単に触れたあと、UMG（アンリアル・モーション・グラフィクス）を使用して「ペーパーニンジャ」のタイトル画面とインゲームのUI制作を実践します。

 LD	 CHR	 ENV	 ANM	 SND	 FX	 TA	 ENG	 UI
<p>推奨職種：(LD)、ENG、UI</p> <p>試作時にリソースの数量を画面に出したり、デバッグ表示を出せるようになっておくのは、レベルデザイナーやゲームデザイナーにとっては必須です。アニメーションやレイアウトをまじめに読み込む必要はありませんが、ハンズオンは一通り済ませておく、手元の試作で困ることはなくなるはずです。</p>								

D.1 UIとワークフロー

ゲームの映像表現に限界のあった時代は、ゲーム画面にはゲーム中の情報が数値としてところ狭しと並べられていましたが、ゲームの映像表現能力が上がっていくにつれ、徐々に過去のものとなっていきました。転換期の先陣を切ったとされるのは、2009年にEAから発売されたシューティングゲーム『Dead Space』です。このゲームでは、HUDに表示することが定番とされたプレイヤーのライフ表示を排除し、ゲーム中にプレイヤーが着込んでいる宇宙服の発光でそれを表現する等々の工夫を、すべてのUI要素に対して行い、情報量はそのままに、ゲーム画面上からUIをほとんど排除しました。

『Dead Space』ほど徹底していないにせよ、近年のゲームのUIは「スマート」の一語につきます。このようなUIを生み出すため、UI関係の仕事の専門性が高まると同時に、ワークフローも大きくその姿を変えてきています。

D.1.1 UIのワークフロー

かつてUIデザインが重要視されていなかった時代にも、ゲーム開発者はUIに皆一言もっており、開発者同士が議論し、研究し、学習し、試行錯誤を行うことで、その時代その時代の最適なUIを作り上げ、この分野の歴史の礎を築き上げました。

しかし、もっとも墮落した開発現場では、①何の知識もなく、学習もしていないゲームデザイナーがUIを方眼紙の上で考え、②2D アーティストが頼まれるままに絵素材を作り、③エンジニアが言われるがままにそれを動かして、画面に乗せて(しかも2週間後くらいになって)ダメだと気付いて①に戻る...という、とんでもない仕事の進め方がまかり通っていたとも聞きます。

UIデザイナーの専門性が認められた現在はどうなっているのでしょうか？ 実は、UIのワークフローは、アニメーションやサウンドよりも早い時期に、デザイナードリブンへ移行しています。これは、**Flash**をゲーム上で再生するソリューションが早い時期から導入されたためです。このWeb用の強力なオーサリングツールにより、UIのデザイナーが、知識と仮説をもとに最適なUIのデザインを自分で行い、自分でアニメーションをつけ、自分でテストし、自分でやりなおすことが可能になりました。

この工程の流れ方は、レベルデザインのグレーボックスに似ています。UIデザイナーが精度の高いモックを使ってUIのデザインを終えると、必要に応じてUIアーティストが絵素材をブラッシュアップします。並行してエンジニアは、オーサリングツール上で動作しているUIと、まったく同じ動作がゲーム上で再現されるように、組み込みを行います。ゲーム側の情報がUI側の表示に利用できるよう**バインド**という紐付けを行うことで、UIがゲーム上で実際に働き始めます。

この工程は、実際に本章で体験することができますが、組み込みなどはプログラムから必要な情報を紐付けるだけの1~2時間未満のちょっとした作業に過ぎません。

D.1.2 UE4におけるUIワークフロー

UE4でも、UIデザイナーがUIの試行錯誤の中心となり、エンジニアがわずかな手間でゲームとUIのタイミングとパラメータを連携させるコードを書くワークフローが基本となっています。

UMG (Unreal Motion Graphics)

UMGは、UE4に標準搭載されたUIのソリューションです。UMGでは、**ウィジェットブループリント**と呼ばれるアセット単位で制作を進めます。

UIデザイナーは、ウィジェットブループリントエディタの**デザイン**編集モードでUI素材を仮想スクリーンである**キャンバス**にドラッグ&ドロップしてレイアウトを作り、アニメーションをつけ、作り込みを進めています。

また、**グラフ**編集モードに切り替えると、ブループリントを使ってUI側にロジックを付け加えることができます。これはFlashにおけるActionScriptのような位置づけになっており、多くの場合はエンジニアが作業することになるでしょうが、ラーニングコストは非常に低く、UIデザイナーが扱うことももちろん可能です。

UMGはUE4の基本機能ということもあって、マルチプラットフォームを意識しており、解像度やアスペクト比の変化を吸収する強力なレイアウト機能を備えています。さらに絵素材やテキストだけでなく、ボタンやチェックボックス、プログレスバーといった、ゲームのメニューでよく使われるインターフェイス部品も標準で備えており、その入出力をゲームと紐付けることもほとんどマウス操作だけで行うことができます。必要なら、エンジニアは普段使っているUE4開発環境をそのまま使って、C++コードを書いて、各UI要素を独自に拡張していくことが可能です。

強いて問題を挙げるとするならば、UMGはUE4独自の機能のため、未経験者はオペレーションを学習する必要があり、UMGで習得したことは他のゲーム開発環境に持ち出すことができない点でしょうか。ワークフローの観点では、UMGと他のソリューションの間にそれほど大きな違いはありませんので、UE4クローズドであることが問題になるとは思えませんが、次に挙げるミドルウェアを共通で使うことで環境の統一を図ることは可能です。

Scaleform GfX

Scaleform GfXは有名なFlash再生ミドルウェアであり、Autodesk社のゲーム用ミドルウェア群Gamewareの製品のひとつです。その歴史は古く、多くのPS3/Xbox 360世代のゲームで使われてきた実績を持ちます。

Flashをオーサリングツールとして用い、ActionScriptをUI側のロジックの実装に使います。また、UMGが備えているマルチプラットフォーム対応や、入出力のUIコンポーネントとほとんど同等なものを「CLIK」というライブラリがサポートしており、機能に死角はありません。

公式ページによると、UE4のインテグレーションは準備中というステータスですが、多くの開発者が望んでいるソリューションであり、遠からず対応が発表されることでしょう。

Coherent UI

UMGはUE4に閉じた環境であり、Scaleform GfXもFlashに閉じた環境ですが、このCoherent UIはオープンな規格であるHTML5に準拠している点が特徴です。もし、読者または読者のチームメイトがHTML5をバリバリ使いこなせるUIデザイナーであれば、その経験をほとんどそのままUIの制御に持ち込むことができます。

とはいえ、Coherent UIのワークフローはUMGやScaleform GfXとほとんど変わることはありません。Coherent UIはUMGデザイナーやFlashのようなオーサリングツールを提供しており、HTML5のことがまったく分からなくてもUIのデザインを進めることができるからです。UI側にロジックを持たせたい場合はJavascriptで作ることができ、ウェブの世界で一般的なHTML5 + Javascriptのコンビネーションはここでも健在です。

A

B

C

D

E

UE4 のインテグレーションはすでにリリースされており、UMG と同じように容易にバインドを行うことができます。独立系開発者向けに特に安い価格が設定されているため、個人開発でも手が届く点は嬉しいところです。詳細は公式ページ¹をご覧ください。

D.2 タイトル画面をデザインする

先の節では、時に画面から UI を排除することも UI の設計となると書きましたが、それでは本書中に解説するものがなくなってしまうので、ここからはあえてベタベタの UI をステップ・バイ・ステップで作っていくことで、UMG のワークフローとバインドの方法を学習しましょう。本節は、オーサリングツール上での作業をまとめており、UI デザイナーと、そのデザイナーをサポートするエンジニアが読むことを想定しています。

初めに下記の手順で、「ペーパーニンジャ」のタイトル画面専用のマップを作ります。

1. メニューバーから [ファイル] > [新規レベル] と操作してダイアログを開き、テンプレートから [空のレベル] を選択します。
2. タイトル画面は、通常のゲームを実行する必要がないので、タイトル用のゲームモードを作成します。[ワールド設定] パネルの、[GameMode] > [GameMode Override] プロパティの右側にある [+] アイコンをクリックします。
3. [新規のゲームモードを作成] のダイアログが出ますので、[コンテンツ > Blueprints] フォルダを指定し、[名前] に「PNTitleMode」と入力して、[OK] ボタンを押します。
4. さらに、[Player Controller Class] プロパティの [+] アイコン (図 D.1) をクリックして、タイトル画面操作用のプレイヤーコントローラを作成しましょう。同じく [コンテンツ > Blueprints] フォルダに、「PNTitlePlayerController」という名前でブループリントを作成します。
5. ツールバーの [保存] ボタンをクリックし、[コンテンツ > Maps] のパスに、「PNTitleMap」という名前で保存します。

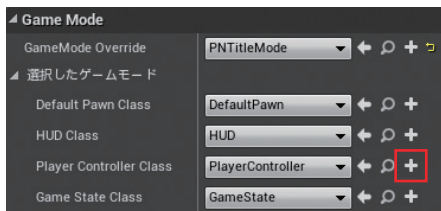


図 D.1 プレイヤーコントローラを作成してセットするショートカット

Note

UI 確認用テストマップ

いま作ったマップは、サンプルゲームのために作っただけであり、新しい UI をデザインするたびにマップを作る必要はありません。ですが、これまでの章でやってきたように、UI のテストのためにマップを作るのは非常に良いやり方です。特に Scaleform Gfx や Coherent UI のような非 UE4 ネイティブのミドルウェアを導入した場合は必須になります。専用テストマップがあれば、オーサリングツール上の動作がゲームで再現できているかどうかをすばやく確認でき、UI をアップデートしたあとのテストも容易になります。

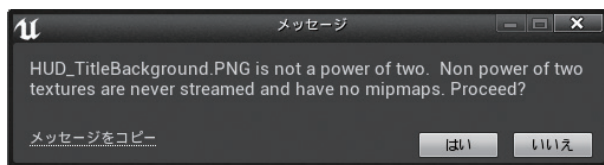
¹ <https://coherent-labs.com/>

D.2.1 ウィジェットブループリントの作成とテクスチャの準備

タイトル画面の下敷きとして使うテクスチャをインポートし、この節で作業を行うUMGのウィジェットブループリントを作って、具体的な作業を行う前の準備をしておきます。

1. コンテンツブラウザのアセットツリーで、[コンテンツ]フォルダの直下に[Textures]という名前のフォルダを作成します。
2. [コンテンツ>Textures]フォルダを選択し、[インポート]ボタンをクリックします。付録データから以下の3ファイルをインポートしてください。
 - PaperNinja_Resource\UI\HUD_TitleBackground.PNG
 - PaperNinja_Resource\UI\HUD_TitleCharaLayer.PNG
 - PaperNinja_Resource\UI\HUD_TitleLogoLayer.PNG

インポートの際に、図D.2のような警告が表示されます。



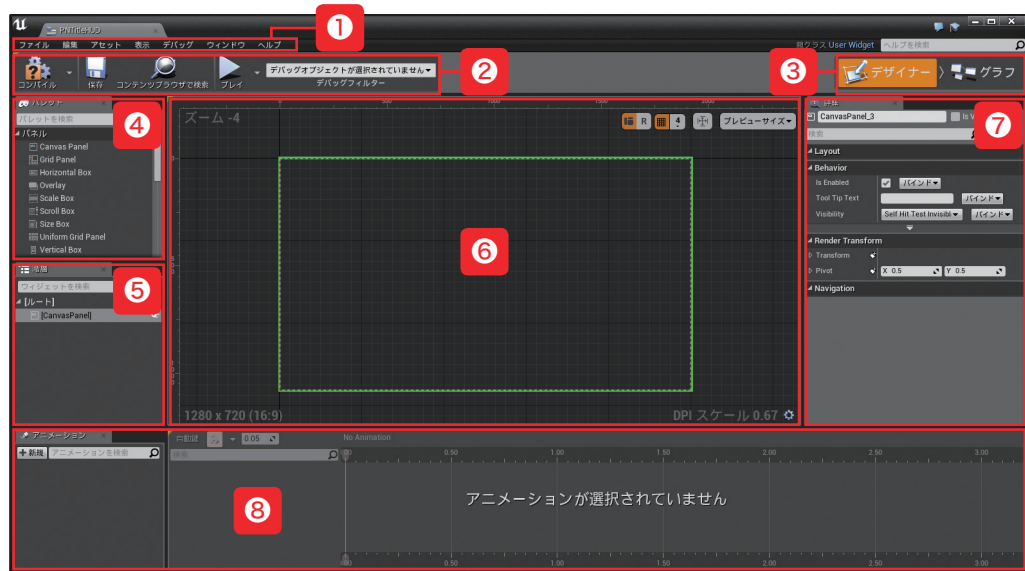
図D.2 2の倍数サイズでないテクスチャに対する警告

UIに使用するテクスチャは2の倍数サイズの正方形とは限らないため、そのようなテクスチャにはストリーム読みが働かず、ミップマップも作られないという内容です。ですが、UIのテクスチャはストリームされてはむしろ困りますし、ミップマップも不要です。何の問題もありませんので、[はい]をクリックして先に進みましょう。

3. インポートした3つのテクスチャをそれぞれテクスチャエディタで開き、[Compression]>[Compression Settings]を[TC_UserInterface2D]に設定します。
4. 次に、UMGのウィジェットブループリントを作りましょう。コンテンツブラウザのアセットツリーで、[コンテンツ>Blueprints]フォルダを選択した状態で、[新規追加]ボタンをクリックし、[ユーザーインターフェイス]>[Widgetブループリント]を選択します。名前を付ける状態になったら、「PNTitleHUD」と名付けます。
5. 作成した[PNTitleHUD]をコンテンツブラウザ上でダブルクリックして、ウィジェットブループリントエディタを開きます。

D.2.2 ウィジェットブループリントエディタ

現在開かれているウィジェットブループリントエディタは、図D.3のようなインターフェイスになっています。



図D.3 ウィジェットブループリントエディタのデザイン編集モード

- ① **メニューバー**：一般的なウィンドウアプリケーションのメニューバーです。
- ② **ツールバー**：よく使う操作を大きなボタンで集めたツールバーです。
- ③ **エディタモード**：UIデザイン編集モードと、UIブループリント編集モードを切り替えるときにクリックします。
- ④ **パレットパネル**：UIデザインで使用する**ウィジェット** (UI部品) が集められており、ここからウィジェットを階層(⑤)やビジュアルデザイナ(⑥)へドラッグ&ドロップすることで、UIのデザインを進めます。
- ⑤ **階層パネル**：ビジュアルデザイナへ配置したウィジェットの親子関係を表示しています。このパネルからウィジェットを選択したり、親子関係を付け替えたりすることができます。
- ⑥ **ビジュアルデザイナ**：現在のレイアウトを視覚的に確認できるビューポートの一種です。ここでウィジェットを直接移動させたり、大きさを変えたりすることもできます。
- ⑦ **詳細パネル**：他のエディタと同様、選択しているウィジェットのプロパティが表示されます。
- ⑧ **アニメーションパネル**：アニメーションに関する作業を行う場所です。D.2.6節で詳しく説明します。

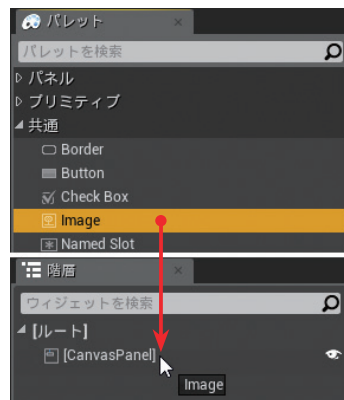
UIデザインの大きな流れを簡単に説明します。ビジュアルデザイナ(⑥)上の緑色の枠(**キャンバス**)が、このエディタにおける仮想的な画面になっています。この枠をレイアウトの目安にしつつ、パレットパネル(④)に集められているウィジェットを、階層パネル(⑤)やビジュアルデザイナ(⑥)にドラッグ&ドロップして、画面にUI部品を追加します。追加した後に、詳細パネル(⑦)でウィジェットの設定を詰めていき、仕上がりの見た目をビジュアルデザイナで確認していきます。

なお、キャンパスのアスペクト比とサイズは16：9の720pになっています。右上の[プレビューサイズ]ボタンを使ってタブレットやスマホの縦画面に変更することができます。さまざまなアスペクト比や解像度の画面に対応したデザインができることがUMGの大きな特徴であり、その際には[プレビューサイズ]を変更して表示テストを行うことになりますが、本書では720pのまま進めることにしましょう。

D.2.3 タイトルの背景を配置する

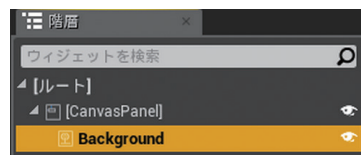
まずはこの枠いっぱいに、D.2.1節でインポートした[HUD_TitleBackground]テクスチャを表示しましょう。

1. [パレット]パネルから、ウィジェットを追加していきます。図D.4のように、[パレット]パネルの[共通]>[Image]をドラッグし、[階層]>[Canvas Panel]へドロップします。



図D.4 [Image]が[CanvasPanel]の子要素になるようにドラッグ&ドロップする

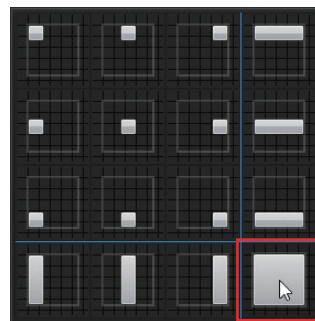
2. ウィジェットに名前をきちんと付けて、作業を進めやすくしましょう。まず[階層]パネル上で、[CanvasPanel]プロパティのツリーを開き、先ほど追加した[Image]ウィジェットをクリックして選択します。そのまま名前を変更する操作を行うか、[詳細]パネルで名前を「Background」に変更します(図D.5)。



図D.5 [階層]パネルで要素を選択し、名前を変更

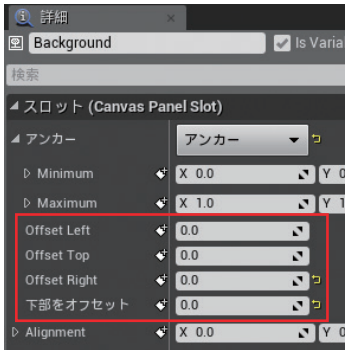
3. [Background] をキャンバスパネルいっぱいに広げましょう。[Background]の位置や大きさをビューポート上で直接変えてもよいのですが、ここは**アンカー**を使用してきっちり位置と大きさを合わせることにします。

[Background]の[詳細]パネルで **アンカー** ボタンをクリックし、一番右下の設定を選択します(図D.6)。



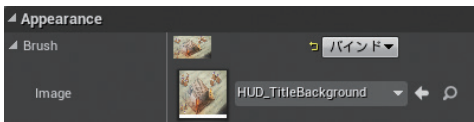
図D.6 [アンカー] ボタンを押して、一番右下の設定を選択

4. ここでは、キャンバスサイズいっぱいにはテクスチャを表示するので、図D.7のように[詳細]パネルですべてのオフセットを「0.0」に設定します。



図D.7 すべてのオフセットを0に設定する

5. 最後に絵をはめましょう。[詳細]パネルの[Appearance]>[Brush]プロパティを開き、[Image]に、先の節でインポートした[HUD_TitleBackground]テクスチャアセットを割り当てます(図D.8)。



図D.8 [HUD_TitleBackground]テクスチャを割り当てる

ビューポートが図D.9のようになっていれば、設定は完了です。

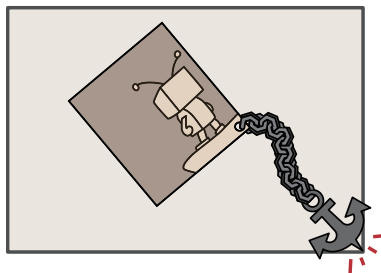


図D.9 ビューポートの状態

Note

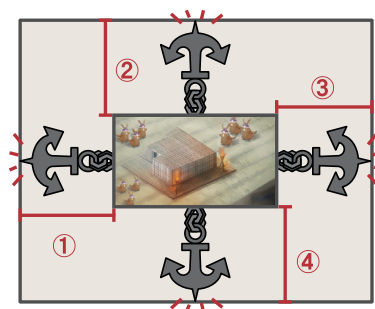
アンカー

アンカーとは、船を水上で固定するときを使う「錨（いかり）」のことで、その役割からレイアウト機能の名前としてよく使われています。図D.10は、アンカーを「右下」に設定してUI部品を配置したときのイメージです。このUI部品は親階層のパネルの右下位置を基準としますので、16：9のフルHD画面でも、9：16のスマホの縦持ちでも、右下に角を合わせるように絵を配置することができます。



図D.10 アンカーのイメージ

本節で設定したアンカーは四辺に対して「錨」を打ったもので、左辺への錨の長さ(図D.11①)が[Offset Left]、上辺への錨の長さ②が[Offset Top]という具合に、4つのオフセット値を設定できます。今回はすべて「0」にしたので、背景画像は親パネルの大きさいっぱいに表示されることになりました。これはスマホの縦持ち画面に出力した場合はアスペクト比がおかしくなりますので、縦持ちと横持ちの両対応を行いたい場合は、上下アンカーに変更するなど、設定の調整が必要です。



図D.11 全方位アンカーの設定イメージ

D.2.4 背景の手前にイメージを重ねる

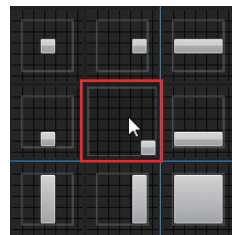
次に、この背景にキャラクターのグラフィックを重ねます。

1. [PNTitleHUD]のウィジェットブループリントエディタに戻り、画像のUI部品を追加しましょう。[パレット]パネルから[共通]>[Image]をドラッグし、[階層]パネルの[CanvasPanel]にドロップします。この新しく追加した[Image]は、先の節で追加した[Background]と同じ階層にあるはずです。
2. 追加した[Image]を[階層]パネルで選択し、[詳細]パネルで名前を「Character」に変更して、分かりやすくしておきます。
3. 今度は先に画像を割り当てましょう。[階層]パネルで[Character]を選択した状態で、[詳細]パネルを[Appearance]>[Brush]>[Image]と辿り、[HUD_TitleCharaLayer]テクスチャを割り当てます。
4. 次にアンカーを設定しましょう。[Character]の[詳細]パネルで

アンカー

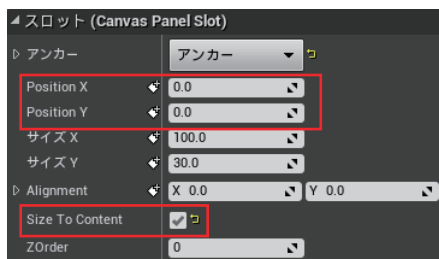
 ボタンをクリックし、今回は図D.12のとおり、右下隅に合わせるアンカー設定を選択します。

これで、画面の形やサイズに関わらず、このテクスチャは画面の右下に角を合わせる形で表示を行う設定になりました。



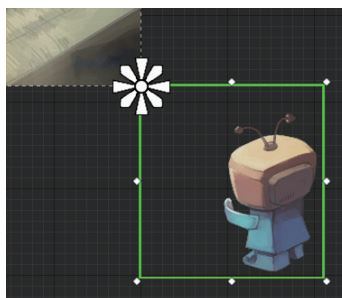
図D.12 親要素の右下隅にアンカーを打つ

5. アンカーを合わせたところで、配置の設定を詰めていきましょう。今回は画面の右下に、絵の右下をびったり合わせる配置にしたいので、[Position X]と[Position Y]を「0.0」に設定します。さらに、UI部品のサイズを元画像のサイズに合わせるために、[Size To Content]にチェックを入れます(図D.13)。



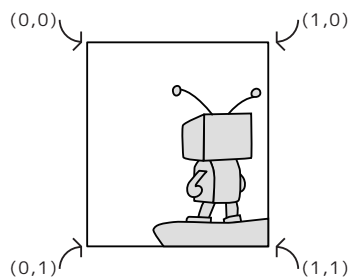
図D.13 キャラレイヤーのUI部品の詳細設定

これで右下角を合わせることができたと思います。や、図D.14のようにおかしいことになってしまいました。実際にはどちらも右下角を合わせたかったのに、親階層の右下隅に、キャラ画像の左上隅を合わせています。「錨」の先っぽは親パネルの右下に投錨しているものの、錨の根元が、画像の左上につながっているため、このようになったのです。



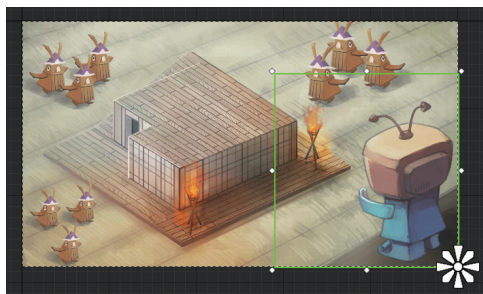
図D.14 キャラ画像の左上隅が、親階層の右下隅にアンカーされた状態

6. 画像のどの位置をアンカーに合わせるのかは、[詳細]パネルの[Alignment (アライメント)]で設定できます(図D.15)。今回の場合は右下に合わせたいので、[Alignment]のXとYともに「1.0」に設定します。



図D.15 [Alignment]の設定値と画像上の位置

7. 画像の位置合わせは意図したとおりにできましたが、今度は[Character]の画像より[Background]の画像が優先して表示されたため、絵が隠れてしまいました。この優先度は[詳細]パネルの[Z Order]で設定します。[Background]の[Z Order]は0ですので、これより大きな数値(ここでは「1」)を打ち込みます。
8. ビューポートが図D.16のようになっていれば完成です。[保存]ボタンを押して保存しておきましょう。

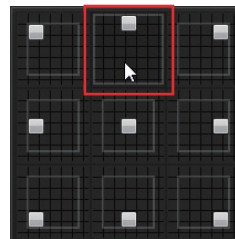


図D.16 画面右下にキャラクターの絵を重ね終えたところ

D.2.5 タイトルロゴを重ねる

前節のアンカーポイントと[Alignment]についての復習を兼ねて、タイトルロゴを画面上部に重ねます。

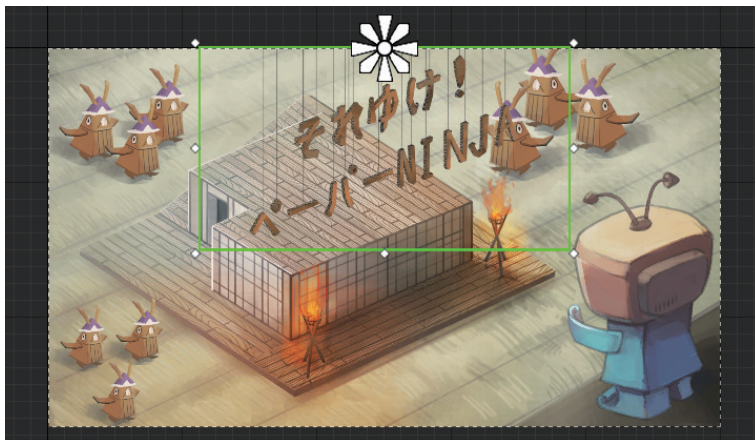
1. [PNTitleHUD]のウィジェットブループリントエディタに戻り、画像のUI部品を追加しましょう。前回と同じように、[パレット]パネルから[共通]>[Image]をドラッグし、[階層]パネルの[CanvasPanel]にドロップします。分かりやすいように、名前を「Logo」に変更しておきます。
2. 今回も先に画像を割り当てましょう。[Logo]の[詳細]パネルで、[HUD_TitleLogLayer]テクスチャを割り当てます。
3. 次にアンカーを設定しましょう。[Logo]の[詳細]パネルで[アンカー]ボタンをクリックし、今回は縦方向は画面（親要素）の上辺、横方向は画面中央に合わせる設定を選択します(図D.17)。



図D.17 タイトルロゴの
[アンカー]設定

4. 狙った画像配置にするために、[詳細]パネルの[スロット]カテゴリのプロパティを詰めます。[Position X]と[Position Y]を「0」、[Alignment]>[X]を「0.5」にセットし、[Size To Content]にチェックを入れます。[Z Order]は「2」としておきましょう。

ビューポートが図D.18のようになっていれば完成です。

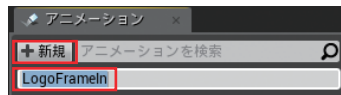


図D.18 タイトルロゴを配置し終えたところ

D.2.6 タイトルロゴをフレームインさせる

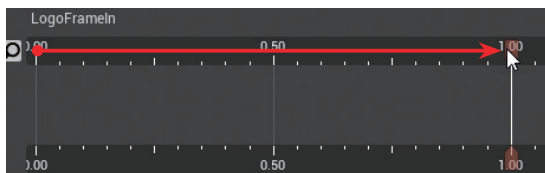
タイトル画面の仕上げに、タイトルロゴにアニメーションをつけてみましょう。

1. アニメーションタブで **+新規** ボタンをクリックしてアニメーションを追加し、分かりやすいように「LogoFrameIn」という名前を付けます(図D.19)



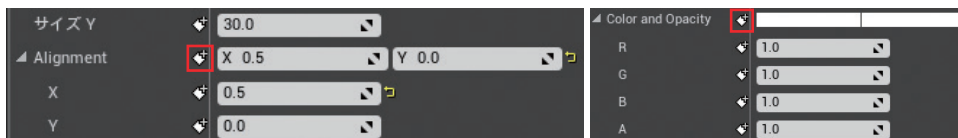
図D.19 アニメーションを追加

2. UMG のアニメーション機能は先にキーを打って動かしてみるとよく理解できます。[アニメーション] パネルで「LogoFrameIn」が選択されていることを確認してから、まず、現在のロゴの位置を最終フレームにするために、タイムラインのカーソルをドラッグして動かして「1.0 秒」の位置に移動します(図D.20)。



図D.20 タイムラインのカーソル位置を移動させる

3. [スロット]>[Alignment]と、[Appearance]>[Color and Opacity (色と透明度)]プロパティの右横にある **+** アイコンをそれぞれクリックします(図D.21)。



図D.21 [Alignment Y]とカラーの透明度をキーに追加する

Note

キー追加ボタン

UMG のタイムラインでは UI 部品の位置や角度だけでなく、[詳細] パネルで **+** アイコンがついているプロパティなら何でもキーとして打つことができます。

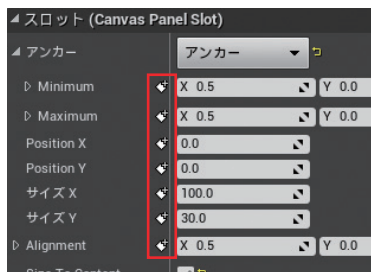
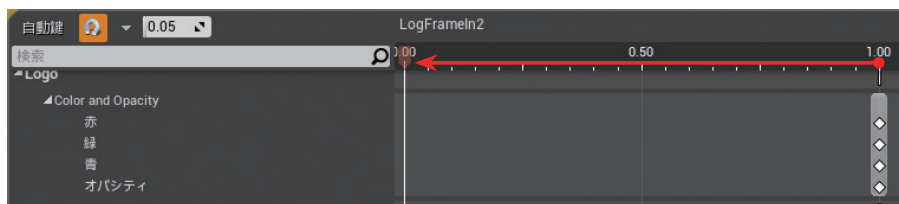




図 D.22 キー追加のアイコンがついているプロパティはすべてアニメーションの対象になる

4. 開始時のキーを打ちましょう。まず、タイムラインのカーソルをドラッグして、「0.0 秒」の位置に移動させます(図D.23)。



図D.23 タイムラインのカーソルを0.0秒の位置へ移動させる


5. 開始フレームにおける、UI 部品のアライメントと透明度を決めましょう。まず開始フレームでタイトルロゴが上部へ確実にフレームアウトしているように、[Alignment]>[Y]を「1.0」に設定します。
これは、D.2.4 節(手順5)で図らずも画像がフレームアウトしてしまった現象を活用したものです。位置を動かしてフレームアウトさせる場合と異なり、画面端のアンカーに対して[Alignment]を動かせば、画面の大きさや画像の大きさによらず確実にフレームアウトを制御することができます。
6. 設定が終わったら、[Y]の左横の  アイコンをクリックしてキーを打ちます。
7. 次に、[Color and Opacity]の[A]を「0.0」に設定して透明にし、 アイコンをクリックしてキーを打ちます。
8. バージョン 4.7 時点のウィジェットブループリントエディタは、プレビュー機能がありません。最低限の確認しかできませんが、タイムラインのカーソルを左右に動かして、思い通りのアニメーションが打てているか確認します。プレビュー機能は今後のバージョンアップに期待しましょう。

D.3 タイトル画面をゲームに組み込む

この節では、D.2節で作ったタイトル画面のウィジェットブループリントを実際のゲームに追加する、いわゆる「組み込み」を行います。どちらかというとなエンジニア向けの内容になりますが、ロジック処理も含めて経験のあるUIデザイナーは次のD.3.1 節だけでも読んでおくと、できることが広がるでしょう。

D.3.1 アニメーションを呼び出す

初めに、ウィジェットブループリント側にロジックを持たせて、呼び出しと同時にタイトルロゴのアニメーションが再生されるように仕込みます。

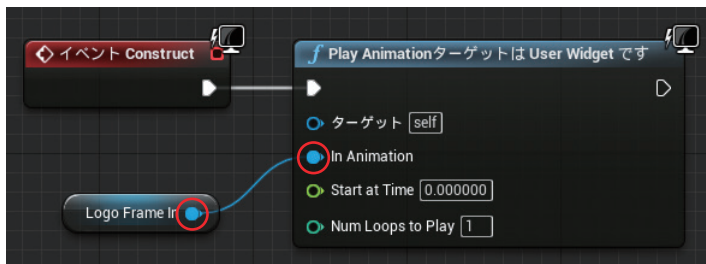
1. ウィジェットブループリントの右上にある[グラフ]アイコン  をクリックして、ワークフローを[グラフ]に切り替えます。
2. イベントグラフ上で [Construct] ノードの実行ピンからワイヤーを伸ばし、アクションリストから [User Interface]>[Animation]>[Play Animation]を選択します。

Note

[Construct] ノード

このウィジェットブループリントが構築される際、つまり、初期化のタイミングで呼び出されるイベントです。

3. [Play Animation]ノードへ[LogoFrameIn]アニメーションを指定しましょう。まず、[マイブループリント]パネルから[変数]>[LogoFrameIn]を、[Ctrl]キーを押しながら、グラフ上にドラッグ&ドロップします。
4. 設置した[Logo Frame In]ノードの出力ピンを、[Play Animation]ノードの[In Animation]入力ピンにつなぎます(図D.24)。



図D.24 再生するアニメーションを指定

5. [コンパイル]を行います。

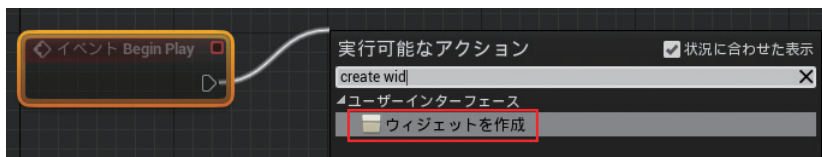
これで、このウィジェットブループリントがランタイムに構築されると、自動的に[LogoFrameIn]アニメーションが再生されるようになります。

D.3.2 レベルブループリントからタイトルHUDを呼び出す

ここまではすべてウィジェットブループリント側の操作でした。ここからはレベルブループリント...つまりゲーム側のロジックを触ります。ゲームが始まると同時に、タイトルHUDを読み込んで画面に表示するところまでを作ります。

ゲームが完成した暁には、この[PNTitleMap]マップが一番最初に読み込まれて、ゲームを起動する予定です。このマップは、3D アクションゲームを遊ぶためのマップではありません。スタートすると同時に全画面を2Dグラフィックスで覆い隠し、タイトル画面を表示するようにしましょう。プレイヤーは何かボタンを押すことで、ステージ1からゲームを開始することができるようになります。

1. レベルブループリントを開きます。
2. [Begin Play]ノードからワイヤーを伸ばし、[ユーザーインターフェイス]>[ウィジェットを作成(Create Widget)]を設置します(図D.25)。



図D.25 [ウィジェットを作成]ノードを設置

Note

[ウィジェットを作成]

[ウィジェットを作成] ノードは、[Class] 入力ピンで指定したウィジェットブループリントを作成する、スポーンに似たノードです。ウィジェットはアクタではないため、作成しただけではマップ上や画面表示に追加されないため、後述する [Add to Viewport] ノードを通じて、ゲームのビューポートに追加する必要があります。



図D.26 [ウィジェットを作成]ノードを設置

3. 設置したノードの[Class]入力ピン右横にあるスロットをクリックして、[PNTITLEHUD]を選択します。
4. [Create Widget]ノードの[Return Value]ピンからワイヤーを伸ばし、[User Interface]>[Viewport]>[Add to Viewport]を選択します。
5. [Create Widget]ノードの次にこのアクションが実行されるように、実行ピンを接続します。最終的に図D.27のようなネットワークになっていれば完成です。



図D.27 ブループリントの完成形

6. [コンパイル]します。

レベルエディタに戻り、ゲームをテストプレイしてみましょう。D.2節で作ったウィジェットが全画面で再生されれば成功です。

Column

アニメーションを増やしたら…


今回の例では、スタートと同時にアニメーションを再生させるという条件だったため、ウィジェットブループリントの [Construct] イベントにアニメーションを再生するロジックを組みました。レベルブループリントからみれば、ウィジェットを初期化するだけで、勝手に [LogoFrameIn] アニメーションが再生された形です。しかし、今後アニメーションが増え、再生タイミングも異なる場合はどうすればよいでしょうか？

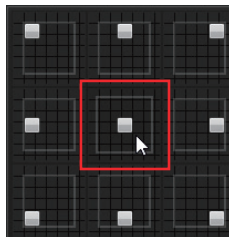
[Play Animation] ノードは外部から呼び出すこともでき、アニメーション変数も公開設定なら外部から取得することができますので、[LogoFrameIn] アニメーションをレベルブループリントのような外部から呼び出して再生することは可能です。しかし、取り回しを考えるとウィジェットブループリント側にアニメーションを再生するためのカスタムイベントを定義し、外部からはそれを呼び出すように作った方がよいでしょう。UMGでUIのアニメーション品質を高めようとなると、タイムラインですべて実装できるわけではないので、外部からは「単なるイベント呼び出し」としておいたほうが、扱いも楽で作り込みもしやすくなるはずです。

D.4 ウィジェットとのインタラクション

タイトル画面の表示はできましたが、ボタンの類がなく、ゲームをスタートさせることができません。そこで簡単なボタンを取り付け、ボタンを押したら何かの処理が実行されるように、ウィジェットブループリント側を改修しましょう。

D.4.1 デザイン作業：タイトル画面にボタンを追加する

1. [PNTitleHUD]のウィジェットブループリントエディタに切り替え、右上の[デザイナ]アイコン  をクリックして、ワークフローを[デザイナ]に切り替えます。
2. [パレット]パネルから[共通]>[Button]をドラッグし、[階層]パネルの[Canvas Panel]にドロップします。分かりやすいように、名前を「StartButton」に変更しておきます。
3. アンカーを設定しましょう。[StartButton]の[詳細]パネルで[アンカー] ボタンをクリックし、親要素の中央にアンカーする設定を選択します(図D.28)。
4. アンカーに合わせて、レイアウトに関するプロパティを設定します。[スロット]>[Position X]と[Position Y]を「0」に、[Alignment]は[X][Y]ともに「0.5」をセットします。また、内容に応じて自動的にサイズを設定する[Size To Content]プロパティにも忘れずにチェックを入れておきます。このボタンには画像を貼り付けたりはしませんが、すぐに役に立ちます。ほか、[ZOrder]を「3」に設定します。
5. ボタンの中にテキストを埋めるために、[パレット]パネルから[共通]>[Text]をドラッグし、[階層]パネルの[StartButton]にドロップします(図D.29)。分かりやすいように、名前を「StartButtonLabel」に変更しておきます。



図D.28 親部品(画面)の中央にアンカーする設定



図D.29 ボタンの中にテキストを入れる

6. [Text] は、文字を流し込んで表示できる UI 部品です。いくつかのプロパティを設定して表示を整えていきましょう。まず、[Content]>[Text]に文字列を入力して、表示する内容を変更します。ここでは「Game Start」としておきます。

次に、[Appearance]>[Font]で文字の大きさを変更します。ここでは「48」にしました。最後に、親階層であるボタンスロットに対して文字列がギリギリにならないように、[Padding (パディング)]プロパティを開いて適度な値を設定します(図D.30)。

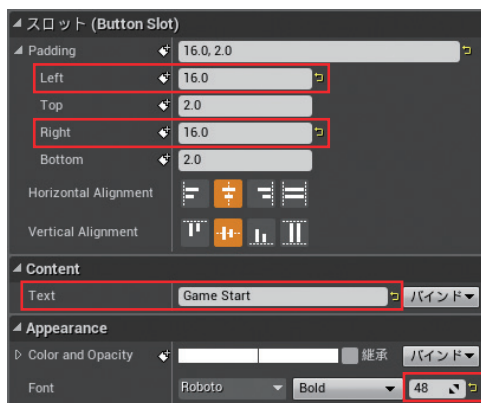


図 D.30 [StartButtonLabel]に対するプロパティ設定

Note

Size To Content

文字の大きさやパディングを設定すると、それに連動して親階層である Button のサイズが自動的に変化していることに気づきましたか？ これは手順 4 で [Size To Content] にチェックを入れたおかげです。[Size To Content] は [Image] スロットのように自身の中身に合わせてサイズを自動決定するものもありますが、子階層のサイズやレイアウトに合わせて自動的にサイズを変更することも基本機能のひとつです。

ビューポートが図 D.31 のようになれば、デザイナー側の作業は完了です。



図 D.31 ボタンを付け終わったところ

D.4.2 組み込み作業: ボタンを押したときの反応を組む

ボックストリガーなどと同じように、ウィジェットブループリントのイベントグラフに、ボタンを押したときに呼び出されるイベントのノードを置くことができます。


1. 現在の[PNTitlePlayerController]の設定では、マウスカーソルが表示されないため、先にそちらを修正しましょう。[コンテンツ>Blueprints]フォルダにある[PNTitlePlayerController]をダブルクリックしてブループリントエディタを開き、[詳細]パネルで[Mouse Interface]>[Show Mouse Cursor]にチェックを入れます。
2. [コンパイル]を行います。
3. [PNTitleHUD] のウィジェットブループリントエディタに戻り、右上のワークフローが[デザイナー]  になっていることを確認します。
4. イベントをグラフに追加しましょう。[階層] パネルで [StartButton] を選択し、[詳細] パネルの [イベント] > [OnClicked] の [プラス] ボタンをクリックします (図 D.32)。
5. [マイブループリント]パネルでイベントディスパッチャーを追加し、「OnGameStarted」と名付けます。
6. [OnGameStarted]イベントディスパッチャーを、イベントノードの近くにドラッグ&ドロップして、[呼び出す]を選択します。[OnClicked]イベントが始まったら、[OnGameStarted]が呼び出されるように実行ピンを接続します (図 D.33)



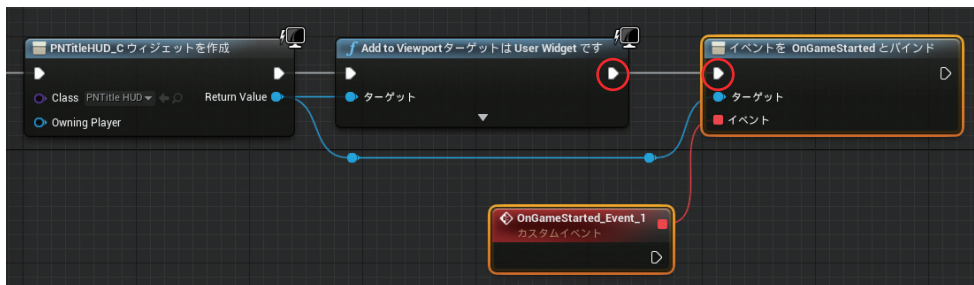
図 D.32 [OnClicked] イベントをグラフに追加する



図D.33 [OnClicked]のネットワーク

7. イベントディスパッチャーを有効にするために、一旦[コンパイル]を行います。
8. レベルブループリント側で、[OnGameStarted] イベントの発行を受け取るネットワークを組みましょう。[PNTitleHUD] のインスタンスは動的に作られたものですが、イベントを**バインド**すれば (18.3.3 節参照)、イベント発行のタイミングを掴むことができます。

レベルブループリントに切り替え、[Create Widget]ノードの[Return Value]ピンからワイヤーを伸ばし、[Default]>[On Game Started を割り当てる]を選択します。配置されたバインドのノードが、[Add to Viewport]ノードの次に実行されるように、図 D.34 のように実行ピンを接続します。



図D.34 イベントのバインド

このように組むことで、ウィジェットブループリントの[OnGameStarted]イベントが呼び出されると、このイベントにバインド(結合)されているカスタムイベント(ここでは「OnGameStarted_Event_1」)と一緒に呼び出されます。

9. まだゲームスタートの遷移を作る段階まで来ていませんので、タイトル画面のボタンが押されたら、デバッグ文字が出るように作っておきましょう。カスタムイベントノードの実行ピンからワイヤーを伸ばし、[Utilities]>[String]>[Print String]を選択します。「Game Start!」と文字が出るようにしましょう(図D.35)。




図D.35 [Print String]ノードをつないだところ

10. [コンパイル]を行います。

レベルエディタに戻り、ゲームをテストプレイしてみましょう。ボタンを押すと、左上にデバッグ文字で「Game Start!」と表示されれば完成です。実際にゲームがスタートするところは、また後の章で作っていくことにしましょう。

D.5 インゲーム HUD をデザインする

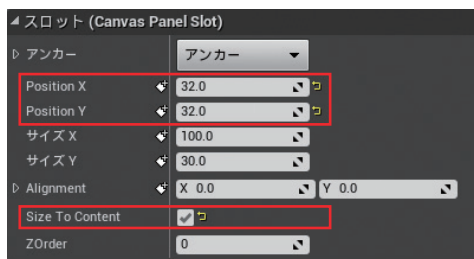
この節では再び UI デザイナにボタンタッチします。タイトル画面を作ることで、ウィジェットブループリントの基本的な使い方を把握することができました。ここからはインゲームのHUDを作り、ゲームプレイヤーが残りのプレイヤーの数...いわゆる**残機数**と、現在持っているコインの数を把握できるようにします。

1. 初めに、必要な残りのテクスチャをすべてインポートしましょう。コンテンツブラウザのアセットツリーで[コンテンツ>Textures]フォルダを選択し、[インポート]ボタンをクリックします。付録データから以下の3ファイルをインポートしてください。
 - PaperNinja_Resource\UI\ICON_Life.PNG
 - PaperNinja_Resource\UI\ICON_Coin.PNG
 - PaperNinja_Resource\UI\ICON_X.PNG
2. インポートした3つのテクスチャをそれぞれテクスチャエディタで開き、[Compression]>[Compression Settings]を[TC_UserInterface2D]に設定します。
3. ウィジェットブループリントを新しく作っていきましょう。[コンテンツ>Blueprints]フォルダを選択した状態で、[新規追加]ボタンをクリックして、[ユーザーインターフェイス]>[Widgetブループリント]を選択します。名前を付ける状態になったら、「PNGameHUD」と名付けます。
4. 作成した[PNGameHUD]をダブルクリックしてウィジェットブループリントエディタを開き、右上のワークフローを[デザイナ]  に切り替えておきます。

この節で行うことは、D.2 節で行ったタイトル画面のデザイン作業で行うこととほとんど変わりませんが、[Horizontal Box]パネルと、[Vertical Box]パネルを利用して、UI 部品をうまくレイアウトしていきます。この節のハンズオンを済ませれば、公式ドキュメントと併せて、パレットにあるほとんどのウィジェットが使いこなせるようになるはずです。

D.5.1 残機数を表示する

1. [パレット]パネルから、[パネル]>[Vertical Box]をドラッグし、[階層]パネルの[Canvas Panel]にドロップします。
2. [Vertical Box]を選択して、[詳細]パネルでアンカー設定を「左上」にして、画面隅から少し距離を取るために、図D.36のように[Position X]と[Position Y]に適当な余白を入れます。ここでも、[Size To Content]プロパティに忘れずにチェックを入れておきます。



図D.36 [Vertical Box]の設定例

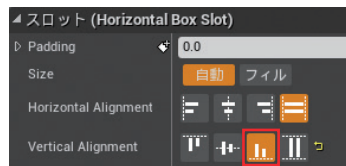
[Vertical Box]は、子階層のスロットを縦に並べる働きをしますが、詳細は次の節で解説します。

3. この[Vertical Box]の内側に、[Horizontal Box]を入れましょう。[パレット]パネルの、[パネル]>[Horizontal Box]をドラッグし、[階層]パネルの[Vertical Box]にドロップします。[Horizontal Box]は、子階層のスロットを横に並べる働きをします。

区別をつけやすくするために、名前を「Horizontal Box For Life」に変更しておきます。

4. 画像を追加しましょう。[パレット]パネルから[共通]>[Image]をドラッグし、[階層]パネルの[Horizontal Box For Life]にドロップします。追加したこのUI部品を選択して、分かりやすいように、名前を「LifeIcon」に変更しておきます。
5. [LifeIcon]の[詳細]パネルで、[Appearance]>[Brush]>[Image]プロパティをクリックして、[ICON_Life]を割り当て、[Image Size]>[X][Y]ともに「100」にして、大きさを調整します。
6. 手順4～5と同じ要領で、[共通]>[Image]を[Horizontal Box For Life]にドラッグ&ドロップし、名前を「X_Icon」に変更して、[Image]プロパティに[ICON_X]を割り当てます。

続けて、[Appearance]>[Image Size]>[X][Y]ともに「50」に変更します。ただし垂直の割り当てを設定する[Vertical Alignment]が[縦方向にフル]設定になっているため、上下に間延びしています。これを[垂直方向下揃え]に変更します(図D.37)



図D.37 [垂直方向下揃え]

7. 画像の右横にテキストを追加しましょう。[パレット]パネルから[共通]>[Text]をドラッグし、[階層]パネルの[Horizontal Box For Life]にドロップします。名前は「NumOfLife」にしておきましょう。

8. テキストボックスが、残機表示の画像とマッチするように、[詳細]パネルで設定を詰めましょう。[Vertical Alignment]は手順6と同様に、[垂直方向下揃え]にし、フォントの大きさは「48」くらいに設定します。[Content]>[Text]には「6」など適当な数値を打ち込んで、実際のゲームに使われるときに近いデザインになるようにしましょう。

ビューポート上の表示が図D.38のようになっていれば、基本的なデザインは完成です。各UI部品のオフセット位置や、パディングを調整して、自分なりのデザインになるようにもう少し詰めてみるのも良いです。



図D.38 残機表示

Note

Boxの中で順番を動かすには

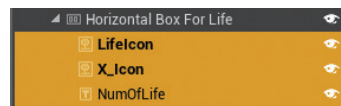
[Horizontal Box]は、子要素を追加された順番ごとに左から並べます。この順番を変えたい場合はビューポート上で矢印のボタンを押して動かします。[Vertical Box]などでも同じ理屈になります。

D.5.2 コイン数を表示する

残機数表示の下に、今度はコインの所持数を表示するためのUI部品を並べていきましょう。レイアウトは同じですので、コピー&ペーストを使って一気に設定することになります。

1. [Verical Box]の内側に、もうひとつ[Horizontal Box]を入れましょう。[パレット]パネルから[パネル]>[Horizontal Box]をドラッグし、[階層]パネルの[Vertical Box]にドロップします。
これでふたつの[Horizontal Box]が、[Verical Box]の働きで縦に並ぶようになりました。追加した[Horizontal Box]の名前を「Horizontal Box For Coin」に変更しておきます。

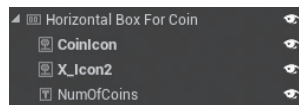
2. [階層]パネルで、[HorizontalBoxForLife]下の3つのUI部品を複数選択し(図D.39)、右クリックして[コピー]を選択します。



図D.39 UI部品を複数選択する

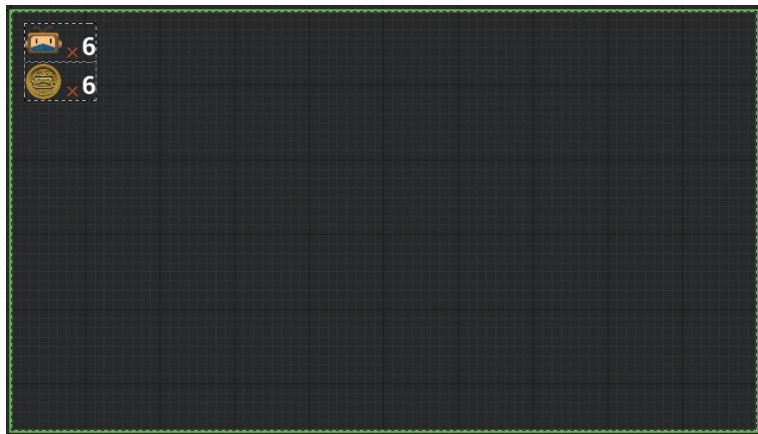
3. [Horizontal Box For Coin]を右クリックして[ペースト]を選択します。この操作により、残機表示のGUIが縦に2つ並ぶようになります。
4. [Horizontal Box For Coin]下の1番上のUI部品をクリックし、[詳細]パネルで名前を「CoinIcon」に変更し、[Image]プロパティに[ICON_Coin]を割り当てます。[Image Size]が元に戻るなので、改めて[X][Y]ともに「100」に設定しましょう。

5. 最後に、2 番目の UI 部品の名前を「X_Icon2」に変更し、3 番目の UI 部品の名前を「NumOfCoins」に変更します(図 D.40)。



図D.40 [Horizontal Box For Coin]
下の UI 部品

ビューポートが図 D.41 のようになっていれば、デザインは完成です。



図D.41 完成したインゲーム HUD のデザイン

D.6 インゲーム HUD をゲームに組み込む

UI デザイナーが [PGameHUD] を作ってくれたので、今度はそれを受け取って組み込むエンジニアの出番です。

D.6.1 PGameMode でインゲーム HUD を追加

まず初めに、ゲーム画面に [PGameHUD] を追加して、表示させるところを済ませてしまいましょう。

1. コンテンツブラウザで、[コンテンツ>Blueprints] フォルダにある [PGameMode] をダブルクリックして、ブループリントエディタを開きます。
2. [Begin Play] ノードのネットワーク終点にあたるバインドノードからワイヤーを伸ばし、[ユーザーインターフェイス]>[ウィジェットを作成 (Create Widget)] を設置します。
3. 設置したノードの [Class] をクリックして、[PGameHUD] を選択します。
4. [Create Widget] ノードの [Return Value] ピンからワイヤーを伸ばし、アクションリストから [Add to Viewport] を選択します。

5. 設置したノードが[Create Widget]ノードの次に実行されるようになっていることを確認し、[コンパイル]します。

ステージ1をロードしてプレイしてみましょう。左上に残機とコイン数を表示するHUDが表示されれば、ここまでの作業は上手く進んでいます。ただし、そこに表示されている数値はウィジェットブループリントで下書きした数値で、ゲーム内の値は反映されていません。ゲーム内の値を、ウィジェットにバインドする作業が必要です。

D.6.2 プロパティのバインディング

1. [PNGameHUD]のウィジェットブループリントエディタに切り替えます。右上のワークフローが[デザイナー]になっていることを確認します。
2. [階層]パネルで[NumOfCoins]をクリックし、[詳細]パネルの[Text]プロパティの右横にある[バインド]ボタンを押して、[バインディングを作成]を選択します(図D.42)。



図D.42 バインディングを作成

バインディングを作成すると、編集モードが[グラフ]に切り替わり、イベントグラフ上に自動的に関数が作成されます(図D.43)。



図D.43 バインド作成により、自動的に作られた関数

この関数の出力値にあたる[Return Value]に書き出した文字列が、[Text]プロパティに書き込まれます。つまり、プレイヤーキャラの持つコイン数を取得し、この[Return Value]ピンに差し込めば、ゲームHUDのコイン数表示はリアルタイムにコインの数を反映したものになります。なお、作成された関数の名前は変更可能です。

3. このHUDに関連づけられたプレイヤーの[Pawn]を取得しましょう。グラフ上の空きスペースを右クリックしてアクションリストを出し、[Player]>[Get Owning Player Pawn]を選択します。
4. 取得した[Pawn]を、プレイヤーキャラである[PNPlayerPawn]までキャストしましょう。手順3で設置したノードの[Return Value]からワイヤーを伸ばし、[ユーティリティ]>[キャストイング]>[PNPlayerPawnにキャストする]を選択します。続けて、このキャストノードの上で右クリックして、[純粹キャストに変更]を選択し、実行ピンを消します。
5. [PNPlayerPawn]からコインの数を取得しましょう。[PNPlayerPawn]キャストノードの[As PNPlayer Pawn]出力ピンからワイヤーを伸ばし、アクションリストから[関数呼び出し]>[Get Num Of Coins]を選択します。

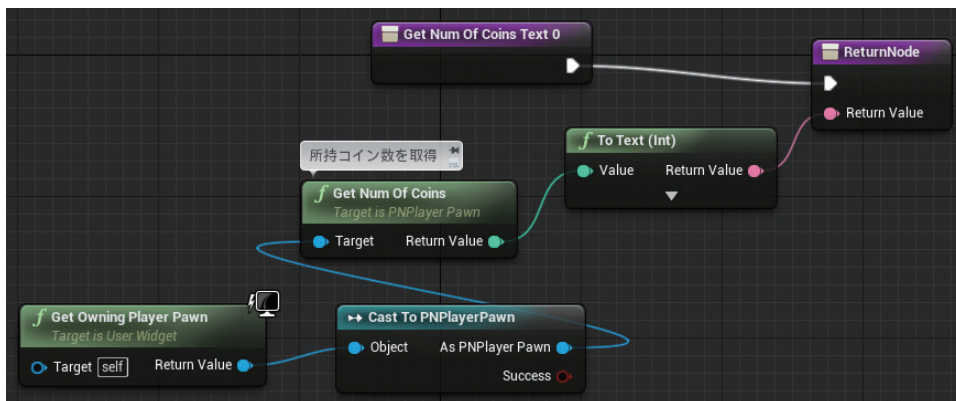
6. [Get Num Of Coins]ノードの[Return Value]ピンを、[Return Node]の[Return Value]入力ピンにつなぎます(図D.44)。



図D.44 [Get Num Of Coins]の戻り値をこの関数の出力値とする

[GetNumOfCoins]は整数値を返す関数ですが、これを文字列のピンにつなぐと、値をコンバートするノードが自動で間に追加されます。

ここまでの作業を行った結果、図D.45のようなネットワークになっていれば、正しく進められています。



図D.45 バインディング関数のネットワークが完成したところ

7. [コンパイル]します。

レベルエディタに戻り、テストプレイしてみましょう。テストには、第18章と第19章を通じて作った[Test_Player_Death]マップが便利です。[C]キーで好きなだけコインを増やすことができ、[K]キーでいつでも死ぬことができます。コインを増やしたときに、コインの数字が増えていけばインゲームHUDの完成です。

本書では、実際には残機数管理を作っていないため、[NumOfLife]のバインドを作ることではできませんが、これは本書を読み終えた皆さんへの宿題としましょう。