





パッケージを作成する

ここまで手順通りに作業を進めてきたみなさんの画面には、UE4 エディタ上でゲームをプレイできる状態が整っていると思います。しかし、この状態のままでは、UE4 をインストールした人にプロジェクトファイルを渡すことでしかゲームで遊んでもらうことができません。本章を理解し、実行ファイルの作り方を覚えることで、他の人にゲームを渡すことができるようになります。

本章では、サンプルゲーム「ペーパーニンジャ」のタイトル画面からゲーム画面につながるシーケンスなど、通して動かすための作業と、Windows OSで動作する実行ファイルを作成する手順を説明します。

								
LD	CHR	ENV	ANM	SND	FX	TA	ENG	UI

推奨職種：(LD)、(CHR)、(ENV)、(ANM)、(SND)、(FX)、(TA)、ENG、(UI)

エディタ上でゲームを動かす場合と、スタンドアロンパッケージでゲームを動かす場合は、完全に同じではありません。プロジェクト後半になってくると、大きな計画に基づくパッケージングスケジュールが動き始めますが、パッケージでしか起きないような調整不足、バグを修正したあと確認できるようになっておくためにも、E.2 節以降に目を通し、自分の手元でパッケージングを行う知識を持っておく必要があります (E.1 節は難しいと思ったら飛ばして構いません)。

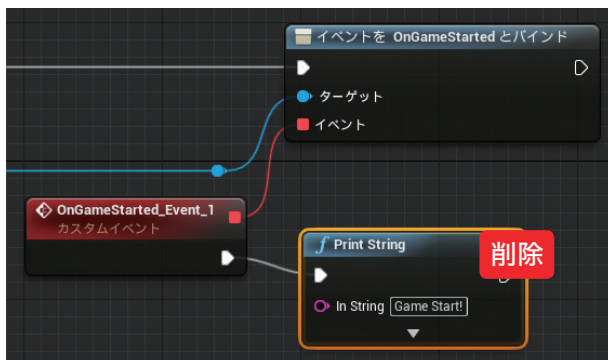
E.1 ゲームを仕上げる

パッケージングの作業に進む前に、Appendix Dで作成したタイトル画面用マップとステージ1のシーケンスをつなげたり、ゲームのちょっとした作り込みを行っておきましょう。また、本編第1刷の出版時に含まれていたエラー(正誤)のうち、重要なものの内容を確認しておきます。

E.1.1 レベル間の遷移

タイトル画面用マップ [PNTTitleMap] でゲームスタートのボタンを押したあと、ステージ1用のマップ [PNStage01_Top]へ遷移する処理を作りましょう。また、タイトル画面のBGMの演奏も仕込んでおきます。

1. レベルエディタで、[コンテンツ>Maps]フォルダの「PNTTitleMap」を開きます。
2. [コンテンツ>Audio]フォルダにある[bgm01_Cue]をマップ上にドラッグ&ドロップしておきます。手抜き(環境音の演奏方法)ではありますが、これでタイトル画面表示の際にジングルを鳴らすことができます。
3. レベルブループリントを開き、[BeginPlay]ノードのネットワーク終点で、[OnGameStarted]イベントとバインドされているカスタムイベントノードのところへ行き、そこにつながっている[Print String]ノードを削除します(図E.1)。



図E.1 [Print String]ノードの削除

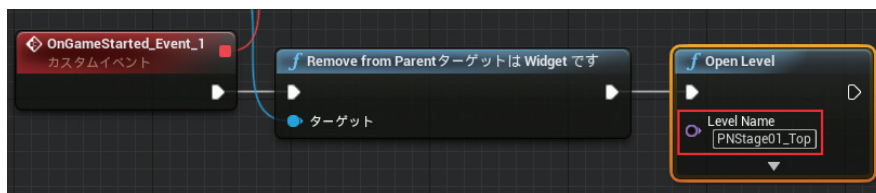
4. [PNTTitleHUD_Cウィジェットを作成]ノードの出力ピンから、もう一本のワイヤーをカスタムイベントの右のあたりまで伸ばして、[Widget]>[Remove from Parent]を選択します。次に、設置されたノードとカスタムイベントの実行ピンをつないでおきます。

Note

[Remove from Parent]ノード

[Remove from Parent] は、ビューポートに追加されたウィジェットを削除する働きがあり、ここでは、[PNTTitleMap]の全画面グラフィックをレベル切り替え前に消去させています。なお、インゲームHUDは常駐させておき、ウィジェットブループリント側の表示コントロールで表示/非表示を実現させたほうがよいでしょう。

5. [Remove from Parent]ノードの実行ピンからワイヤーを伸ばし、[Game]>[Open Level]を選択します。設置された[Open Level]ノードの[Level Name]入力ピンに「PNStage01_Top」と入力します(図E.2)。



図E.2 [Open Level]ノード

6. [コンパイル]します。

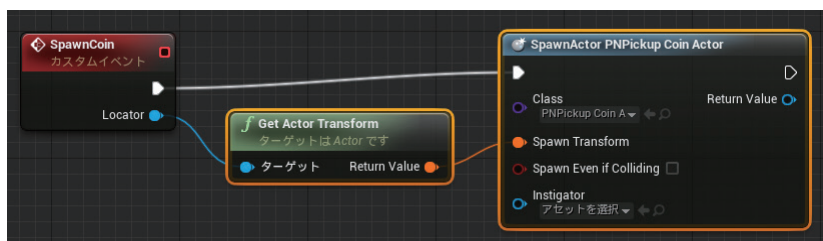
レベルエディタに戻ってゲームをテストプレイしてみましょう。タイトル画面で[Game Start]ボタンをクリックしたときに、「ステージ1」からゲームプレイが始まればうまくできています。

E.1.2 スイッチを踏んだらコインを出現させる

ステージ1にちょっとした仕込みを追加しましょう。現状で2階の床スイッチを踏むと、巻物へと続くふすまが開きます。さらに、この床スイッチは5秒で自動的にオフとなり、ふすまも閉まってしまいます。しかし、5秒以内にふすまの先に進むのは、それほど難しいことではありません。

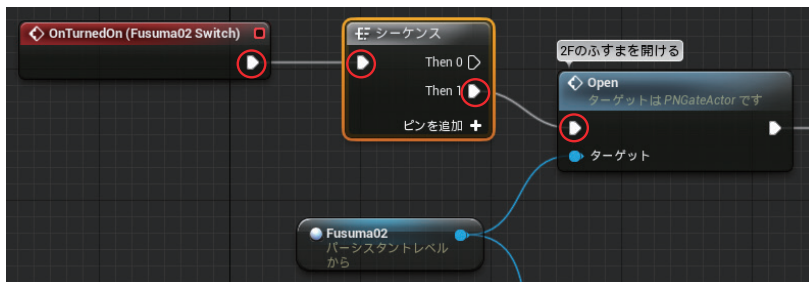
そこで、スイッチを踏んだときに、単にふすまが開くだけでなく、スロープでコインがスポーンするような仕込みを入れ、プレイヤーにちょっとした判断を迫るようにしてみましょう。

1. コンテンツブラウザで、[コンテンツ>Maps>Stage01]フォルダに入っている[PNStage01_Top]をダブルクリックして開きます。
2. [レベル]パネルで[PNStage01_GB]をダブルクリックして、[現在のレベル]を変更します。
3. スロープに4つほど[ターゲットポイント]を配置します。
4. ツールバーを[ブループリント]>[サブレベル]>[PNStage01_GBを編集]と操作するか、[レベル]パネルの[PNStage01_GB]の並びにあるゲームパッドアイコンをクリックして、このサブレベルのブループリントエディタを開きます。
5. **レベルブループリントでもサブルーチンの作成が可能です。** イベントグラフの空きスペースに新しいカスタムイベントを作り、「SpawnCoin」と名付けます。
6. [SpawnCoin]ノードを選択している状態で、[詳細]パネルから入力値を1つ追加し「Locator」と名付け、その種類は[オブジェクトリファレンス]>[Actor]とします。
7. 本編 18.1.1 節の要領で、[Locator]出力ピンの情報をロケーターに使用して、[PNPickupCoinActor]をスポーンするノードネットワークを、[SpawnCoin]ノードの続きに設置します(図E.3)。



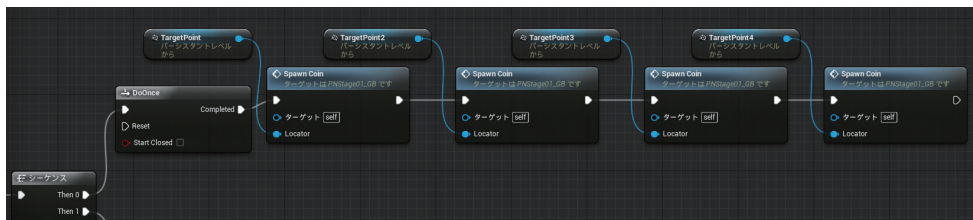
図E.3 [Locator]を利用してコインをスポーンする

8. [Fusuma02 Switch]の[OnTurnedOn]イベントノードの近くの空きスペースを右クリックして、[ユーティリティ]>[フロー制御]>[シーケンス]ノードを設置します。このノードの解説は本編20.5.3節「シーケンスを使う」(p.350)をご覧ください。
9. イベントノードとシーケンスノードの実行ピンをつなぎ、さらに、シーケンスノードの [Then 1] ピンを [Open] ノードにつなぎます (図E.4)。



図E.4 シーケンスノードを使用して流れを整理する

10. [Then 0]ピンからワイヤーを伸ばし、[Utilities]>[Flow Control]>[DoOnce]ノードを設置します。
11. 設置したノードの続きに、4つの[関数呼び出し]>[SpawnCoin]ノードをつなぎます。続けて、各[SpawnCoin]ノードの[Locator]入力ピンに、手順3で設置したターゲットポイントアクタのリファレンスをつなぎます (図E.5)。図は拡大してご覧ください。



図E.5 [SpawnCoin]の呼び出し

12. [コンパイル]します。

レベルエディタに戻り、ステージ1をテストプレイしましょう。床スイッチを踏んだときに扉が開き、コインが4枚登場すればうまくいっています。コインを取ったあと、もう一度床スイッチを踏んでも、何も起きないことを確認しておきましょう。

Column

コインのスパーナーを作る

この章まで読み進めた読者の方であれば、別のステージでも使う可能性のある仕組みについては、レベルブループリントではなく、専用スパーナーを準備した方が何かと便利だと気づくはず。Appendix Aの解説を参考に、専用スパーナーの作成にも挑戦してみましょう。[DoOnce]もスパーナー側に持つことができますし、イベントディスパッチャーと格闘すれば、出現後x秒で消えるが、コインを取っていなければもう一度スポンさせることができる……といった仕組みも組み込めます。

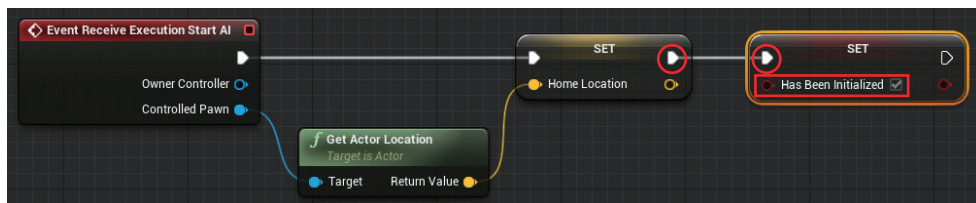
E.1.3 敵AIのデコレーターを確実にする

第23章で作成した独自のデコレーター「DistanceLimit」は、「今回追跡を開始した地点から、指定距離内であれば行動を続ける」デコレーターではなく、「初めて追跡を開始した地点から、指定距離内であれば行動を続ける」デコレーターになっていました。ステージ1ではあまり支障はありませんが、他のステージを作る際にかなりきつい足かせとなりますので、この節で完全なものへ仕上げておきましょう。

第23章の時点では、このデコレーターは実行開始イベント([Receive Execution Start AI])時に現在のアクタの位置を変数[HomeLocation]に控えておき、実行継続チェック([PerformConditionCheckAI])時に[HomeLocation]と最新の位置の間の距離を調べて、継続の可否を決めるという考え方で作られていました。つまり、実行開始イベント→実行継続チェックの順番で呼び出される前提で作られていたのですが、実際には、実行継続チェック→実行開始イベントという順番で呼び出されることもあります。

そこで、本編では使用しなかった実行終了イベント([Receive Execution Finish AI])をイベントグラフに追加し、[Home Location]が正しく初期化されている状態かどうかをフラグで管理します。初期化されていない場合は、実行継続チェックを無条件で通過するように作りかえましょう。

1. [コンテンツ>AI]フォルダに格納されている[PNBTD_DistanceLimit]をダブルクリックして、ブループリントエディタを開き、[EventGraph]タブを押します。
2. [マイブループリント]パネルで新しい変数を追加し、「HasBeenInitialized」と名付け、[変数の種類]を「Boolean」にしておきます。
3. [Receive Execution Start AI]ノードの終点の続きに、変数[HasBeenInitialized]の[SET]ノードを配置し、この変数がオン(True)になるように入力ピンにチェックを入れておきます(図E.6)。



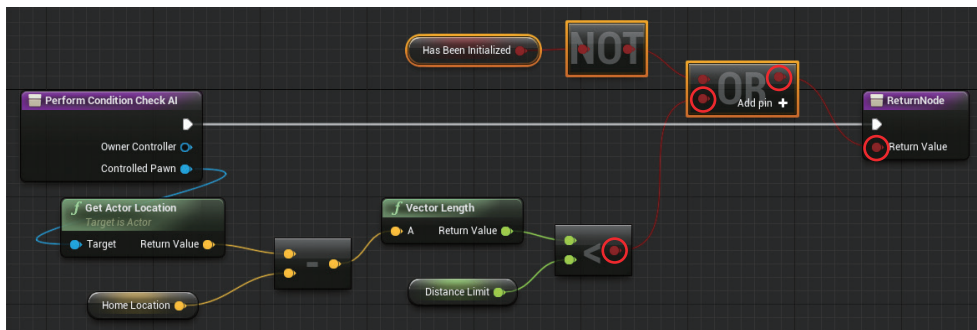
図E.6 [Execution Start AI]ノードネットワークを改良する

4. [Execution Start AI]ノードの下空きスペースを右クリックして、[イベントを追加]>[AI]>[Receive Execution Finish AI]を選択します。
5. 設置した[Execution Finish AI]ノードの続きに、変数[HasBeenInitialized]の[SET]ノードを配置し、この変数がオン(True)になるように入力ピンにチェックを入れておきます(図E.7)。



図E.7 [Execution Finish AI]のノードネットワーク

6. [マイブループリント]の[関数]>[PerformConditionCheckAI]をダブルクリックして、当該の関数グラフを開きます。
7. [マイブループリント]パネルから[変数]>[HasBeenInitialized]の参照([GET])ノードを配置しましょう。[Ctrl]キーを押しながらグラフ上にドラッグ&ドロップするか、普通にドラッグ&ドロップしたあとサブメニューから[取得]を選択します。
8. [HasBeenInitialized]ノードからワイヤーを伸ばし、[Math]>[Boolean]>[NOT Boolean] (詳細は本編p.440 参照)を選択します。
9. [NOT Boolean]ノードからワイヤーを伸ばし、[Math]>[Boolean]>[OR Boolean]を選択します(詳細は本編p.323 参照)。
10. 本編で設置済みの[<]ノードの出力ピンからワイヤーを伸ばし、先ほど設置した[OR Boolean]ノードのBピン(下側のピン)につなぎ、[OR Boolean]ノードの出力ピンと[Return Node]の入力ピンをつなぎます。作業の要点は、図E.8を拡大してご覧ください(本節で追加したノードはハイライトしてあります)。



図E.8 [Perform Condition Check AI]のノードネットワークを改良したところ

11. [コンパイル]します。

これをテストするには、[Test_AI_Patrol]マップを使うとよいでしょう。

Tip

パラメータ調整

今回の修正に合わせて、ビヘイビアツリー上の[Distance Limit]設定値も少し調整しておくことをお勧めします。「300」よりは「500」程度にしておいたほうが、敵との追いかけても盛り上がるでしょう。

E.1.4 エラッタ：ふすまが正しく影を落とすようにする

「Unreal Engine 4で極めるゲーム開発」の本編第1刷には多くの間違いが含まれており、その内容は正誤表という形で(株)ボーナデジタル社の書籍サポートページ¹にまとめられています。大変申し訳ありません。正誤表の中には、作業手順にちょっとした混乱をもたらす表記ミスもあれば、ゲームの一部が正しく動作しない「バグ」に相当するものもあります。本節と次のE.1.5節では、正誤表を読まずにここまでできてしまった方のために、正誤情報の中でも特にクリティカルな部分(後者の「バグ」の部分)を紹介し、問題が起きない状態へと仕上げていきます。なお、ここまで「**正誤表**」を参照しながら作業を進めてこられた方や、「**第2刷(書籍最終ページの奥付にてご確認いただけます)**」で進めてこられた方は、すでに問題が解決されているはずですので、本節と次節を読み飛ばしていただいても問題ありません。

第28章では、スタティックなライトが、ムーバブルなアクタに対して動的なシャドウを落とさない仕様になっている点について言及がなく、手順通り進めると、2階部のふすまの影が一部落ちない現象を起こしていました。まずは、この問題を修正していきましょう。

1. 引き続き[PNStage01_Top]を読み込んだ状態で作業を続けます。
2. [レベル]パネルで[PNStage01_Light]のロックを外し、ダブルクリックして[現在のレベル]に設定します。ほかのサブレベルを間違えて編集しないように、代わりにロックしておきましょう。
3. [ワールドアウトライナ]パネルで[Light and Fog]>[CenterPointLight]を選択し、[詳細]パネルで[可動性]を[ステーションナリー]に変更します。
4. 同様に、[Light and Fog]>[FirePointLight]を選択し、[詳細]パネルで[可動性]を[ステーションナリー]に変更します。
5. [コンテンツ>Blueprints]フォルダの[PNBonfireActor]をダブルクリックしてブループリントエディタを開き、[コンポーネント]パネルで[PointLight]コンポーネントを選択し、[詳細]パネルから[可動性]を[ステーションナリー]に変更します。
6. レベルエディタに戻り、[ビルド]ボタンの右脇の[▼]ボタンを押して、サブメニューから[ライティングのみビルド]を選択します。

ゲームをテストプレイしてみましょう。ふすまの可動部にも影が落ちるようになっているはずです。

E.1.5 エラッタ：効果音が確実に聴こえるようにする

Appendix Cで再確認したように、カメラはプレイヤーキャラから1200ユニット離れており、減衰設定[PN_AT_Default]の[Falloff Distance]で設定される可聴距離「1000」を僅かに上回っています。これは調整のしどころではありますが、現在の設定ですと、ふすまの音など、プレイヤーキャラの位置から離れた場所で使われるサウンドエフェクトが可聴距離の外に出てしまうケースがあります。そこで、この値の距離を伸ばしましょう。

1. [コンテンツ>Audio]フォルダの[PN_AT_Default]アセットをダブルクリックして編集画面を開きます。
2. [Settings]>[Falloff Distance]を「2000」に設定します。

[プレイ]ボタンを押して、ゲームをテストプレイしてみましょう。2階に進み、床スイッチを手前、ふすまを奥にしたカメラ位置で床スイッチを踏んで、ふすまの音が聞こえればうまくいっています。

1 <http://www.borndigital.co.jp/book/support/5436.html>

E.1.6 スクリプトの警告を取る

ブループリントに実行不可能な重大なエラーがあった場合は、コンパイルエラーが発生するため、ゲームを実行する前に原因を調査して解決することができます。しかし、ロジックのミスの中には実行してみるまで分からないものもあり、その中にはさらに、(非技術者にとって)「思い通り動いているのに、なぜか警告される」類のものもあります。私たちはこれまでにひとつの実行中スクリプトエラーに目をつむって開発を続けてきました。このエラーは非常に典型的なエラーですので、本節では、それを例に取り修正を行います。

Column

修正はエンジニアへ

チームで開発しており、チームメンバーにエンジニアがいるのであれば、(試行錯誤が終わったスクリプトなら)この手の警告の修正はエンジニアに依頼して修正して貰うのが一番です。その際に**試行錯誤の終わり具合**や、テストマップも伝えて、クリンナップしてもらってもいいでしょう。エンジニアがスクリプトを修正すると、非技術者が続きを引き取りにくい仕上がりになることもあるので、自分の中で固まっていない部分や調整が残っている箇所をしっかりと伝えて、良い形で修正して貰うようにしましょう。

中にはこの種の作業やコミュニケーションを嫌がるエンジニアもいるかもしれませんが。若いエンジニアの方は、出し抜くチャンスだと考えて積極的に取り組むことをお勧めします。この種のやり取りは、机上で習ったエンジニアリングと、コンテンツ制作で活きる技術力のちょっとした違いを感じる貴重な機会となるでしょう。

1. [コンテンツ>Maps>Stage01]フォルダの[PNStage01_Top]を開いて、[プレイ]ボタンを押してゲームをプレイし、終了させます。画面右下に図E.9のような表示が差し込まれますので、[メッセージログを表示する]のリンクをクリックします。

🚫 エディタで再生中にエラー/警告が報告されました
メッセージログを表示する

図E.9 ゲームのエラー/警告報告

2. エラーレポートの中に図E.10の警告「Accessed None 'CallFunc_BreakHitResult_HitActor' from node ReturnNode in blueprint PNPawn_Rabbit」があることを確認し、[ReturnNode]のリンクをクリックします。するとこのエラーの発生源である [PNPawn_Rabbit] ブループリントの [CanSeePlayerPawn]関数グラフが自動的に開きます。

```

🚫 Accessed None 'CallFunc_BreakHitResult_HitActor' from node ReturnNode in blueprint PNPawn_Rabbit
🚫 Accessed None 'CallFunc_BreakHitResult_HitActor' from node ReturnNode in blueprint PNPawn_Rabbit
🚫 Accessed None 'CallFunc_BreakHitResult_HitActor' from node ReturnNode in blueprint PNPawn_Rabbit
🚫 Accessed None 'CallFunc_BreakHitResult_HitActor' from node ReturnNode in blueprint PNPawn_Rabbit
🚫 Accessed None 'CallFunc_BreakHitResult_HitActor' from node ReturnNode in blueprint PNPawn_Rabbit
🚫 Accessed None 'CallFunc_BreakHitResult_HitActor' from node ReturnNode in blueprint PNPawn_Rabbit

```

図E.10 実際のエラー

Note Accessed Noneエラー

本編～付録と通じて使ってきた「ブーリアン(Boolean)」「整数(Integer)」「浮動小数(Float)」「ベクトル(Vector)」「ローテータ(Rotator)」といった変数の種類には、必ず何らかの値が入っています。例えば、「所持金ゼロ」のとき、人はそれを「お金がない」と言いますが、数値の観点ではそこに「0」という値が入っています。一方、「オブジェクト(Object)」変数や、値が「無(None)」ということがありえます。例えば、「視線の先にあったアクタ」を示す変数は、時に「敵キャラ」、時に「壁」、時に「アイテム」を格納しますが、視線の先にアクタがなければその値は無(None)になることもあります。

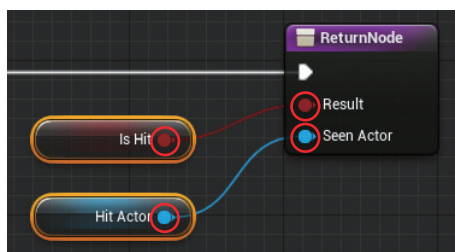
今回の警告は、[Trace] の際にヒットが得られなかった場合([Return Value] がオフ(False) の場合) でも、[OutHit] を Break してそれを [Return Node] につなげているのが原因です。

- 修正方法にはいくつか方法がありますが、今回は本編でも扱わなかった概念の紹介のために、**ローカル変数**を用いた修正を行っていきます。[マイブループリント]パネルの[ローカル変数]グループの右側にある[+] ボタンにマウスカーソルを重ねてクリックし、新しいローカル変数を追加します。追加されたローカル変数に「IsHit」と名付け、変数の種類を[Boolean]に設定します。

Note ローカル変数

地方ローカル局のテレビ番組は、その地方でしか視聴することができません。同様に(?)、ローカル変数は、**関数グラフごと**に作ることができる特殊な変数であり、当該の関数グラフの中だけで読み書きが可能です。関数グラフの中で値を一旦どこかに保存しておいて、あとでまとめて使いたい場合（まさに今回のケース）などに便利に使用できます。

- 同様にして新しいローカル変数を追加し、「HitActor」と名付け、変数の種類を[オブジェクトリファレンス]>[Actor]に設定します。
- [ReturnNode]の近くに、ローカル変数[IsHit]と[HitActor]をドラッグ&ドロップして参照([GET])ノードを設置し、それぞれ[ReturnNode]の[Result]入力ピンと[SeenActor]入力ピンに接続します(図E.11)。



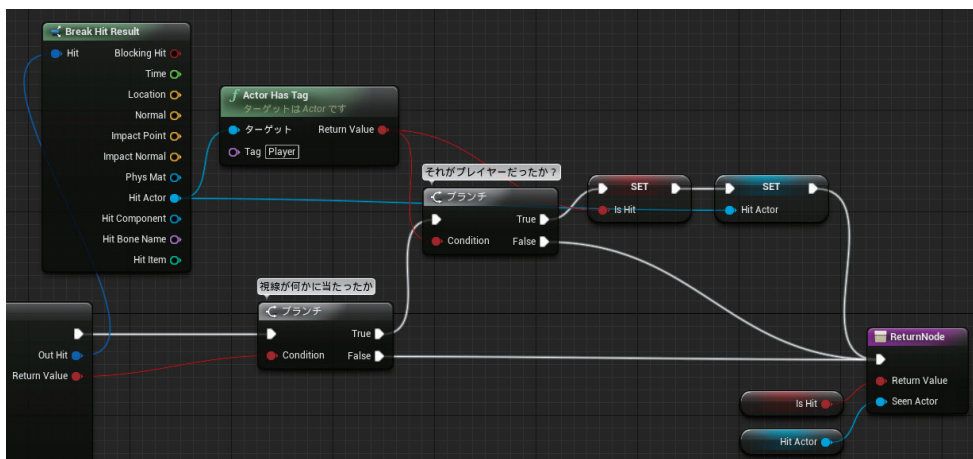
図E.11 ローカル変数を[ReturnNode]に接続

6. [Line Trace by Channel]ノード(以下[Line Trace]ノード)の実行出力ピンを、[Alt]キーを押しながらクリックして、ワイヤーを切断します。
7. [Line Trace]ノードの[ReturnValue]出力ピンからワイヤーを伸ばし、[ユーティリティ]>[制御フロー]>[ブランチ]ノードを設置して、[Line Trace]ノードの実行出力ピンとつなぎます。解説上、このブランチノードを「ブランチ①」と呼ぶことにします。視線チェックでどのアクタにもヒットしなかった場合は、[ReturnValue]は[False]になるはずなので、「ブランチ①」ノードの[False]とReturnNodeをつなぎます(図E.12)。



図E.12 一段目のブランチ

8. 次に、[Actor Has Tag]ノードの出力ピンからワイヤーを伸ばし、先の手順と同様にブランチノードを設置します(これを解説上「ブランチ②」とします)。「ブランチ①」の[True]側と「ブランチ②」をつなぎます。また「ブランチ②」の[False]側を[Return Node]とつなげておきます。
9. 「ブランチ①」と「ブランチ②」の両方のチェックを通過した先(つまり「ブランチ②」の[True]側)に、図E.13のようにローカル変数[IsHit]と[HitActor]の代入([SET])ノードを設置して、図と同様にワイヤーをつなぎます。



図E.13 修正した[CanSeePlayerPawn]

10. [コンパイル]します。

ゲームをプレイして、警告が消えたことを確認しましょう。今回のノードネットワークを改めてチェックすると、[ブランチ①]を[True]に抜けた時点で[Line Trace]ノードの[OutHit]が[None]ではないことが確定します。その先で[Break Hit Result]の出力ピンの情報を使っても、エラーではなくなります。

Tip**UE4.9以降**

UE4.9以降であれば、[ReturnNode]を複数配置できるようになっているため、ローカル変数を使わないノードネットワークを組むことも可能です。ここは難しいところ（議論にもなるところ）ですが、スクリプト言語の仕様次第では、[ReturnNode]が1つのほうが複雑性が下がり、ヒューマンエラーも減って、スクリプトも読みやすくなります。しかし、かえてそれにこだわると、スクリプト構造の専門性が高まり、非技術者にとって扱いにくいものになることもあるため、UE4のバージョンアップに合わせて、自分やチームにとって一番良い形を見つけていくことが重要になります。

Note**[AND Boolean]と[OR Boolean]の挙動**

エンジニアの方の中には、今回の修正はブランチを使わなくても、論理演算と三項演算を使えば実行ラインを一本にまとめることが可能ではないかと思った方もおられるでしょう。しかし、UE4.7時点での[AND Boolean]と[OR Boolean]は、C++の論理演算とは異なる挙動をします。例えば、[AND Boolean]はAピンがFalseだった場合、Bピンを評価する必要はありませんが、実際にはBピンの評価は実行されます。AピンとBピンの内容を確定させてから、論理演算を実行しているのです。同様に[OR Boolean]はAピンがTrueだった場合は、Bピンを評価する必要はありませんが、[AND Boolean]同様、評価が実行されてしまいます。

これが言語仕様というなら納得できないことはありませんが、Noneアクセスエラーを消すときにノードネットワークがにわかに複雑化するのをみると、少々歯がゆい気分です。

E.2 パッケージ化の設定と実行

コンテンツをプラットフォームに最適化されたフォーマットに変換し、ブループリントをビルドして実行ファイルを作成することを、「パッケージ化」もしくは「ビルド」と呼びます。この作業を行うことでUE4エディタがインストールされていない環境でもゲームが動作するようになります。

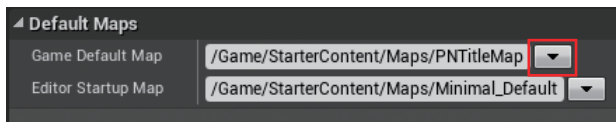
手順自体はとても簡単ですが、若干準備も必要です。以下の手順で実際にパッケージ化を体験してみましょう。

E.2.1 ゲーム開始時のレベルを指定する

アンリアルエディタ上では、好きなレベルを読み込んで[プレイ]ボタンを押してゲームをスタートさせることができますが、実際のゲームでは使用するレベルの順番というものがあります。「ペーパーニンジャ」で言えば、一番最初に読み込んで欲しいのは「PNTITLEMAP」になります。その指定を[プロジェクト設定]で行っておきます。

1. ツールバーの[設定]ボタンを押して、サブメニューから[プロジェクト設定]を選択します。
2. [プロジェクト設定]ウィンドウが開いたら、左側のメニューから[プロジェクト]>[マップ&モード]をクリックします。

3. [Default Maps]>[Game Default Map]プロパティに、ゲームを起動した直後に読み込むマップを指定します。今回のゲームでは、タイトル画面の役割を果たす[PNTitleMap]になります。[▼]ボタンをクリックして、リストから[PNTitleMap]を選択します(図E.14)。

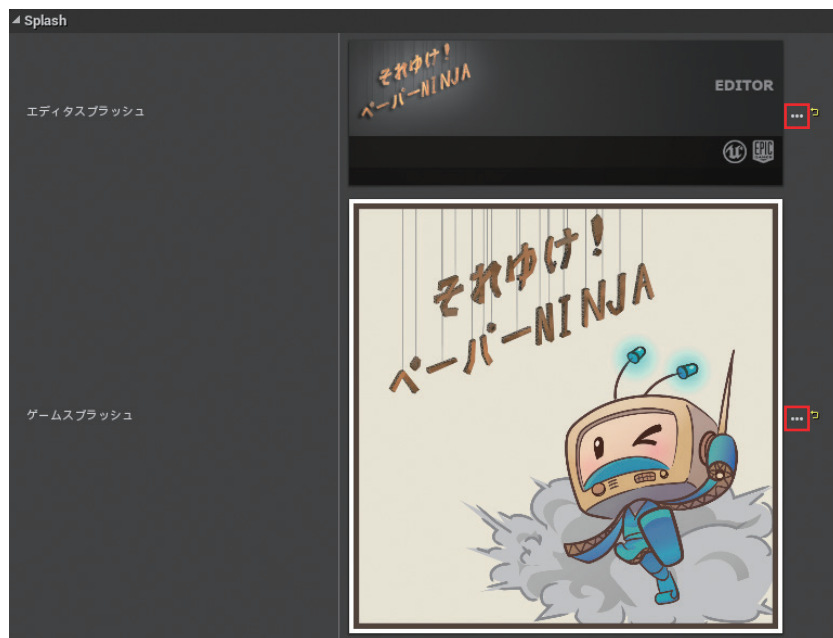


図E.14 ゲーム起動後に読み込むマップを指定

E.2.2 Windows プラットフォームのビルド設定

本章では、Windows 向けにビルドを行いますので、その設定方法もみていきましょう。

1. プロジェクト設定ウィンドウの左側のメニューから[プラットフォーム]>[Windows]をクリックします。
2. [Splash]>[ゲームスプラッシュ]の右横にある[...]ボタンをクリックして、画像ファイルを登録します(図E.15)。ここでは、付録データの[PaperNinja_Resource\BuildResources\GameSplash.BMP]を選択してください。



図E.15 スプラッシュ画像の変更

Note エディタスプラッシュ

今更ですが、付録データには「エディタスプラッシュ」の画像も用意してあります。気が向いたら、エディタスプラッシュを[PaperNinja_Resource\BuildResources\EditorSplash.BMP]に変更してみましょう。こちらはエディタを起動する際のスプラッシュになります。エディタスプラッシュを変更すると、「UE4 ツールを起動している」というより「自分たちのゲームを作っている」という感覚が高まるので、開発序盤に変更し、以降定期的に差し替えることをお勧めします。

Note スプラッシュ画像のフォーマット

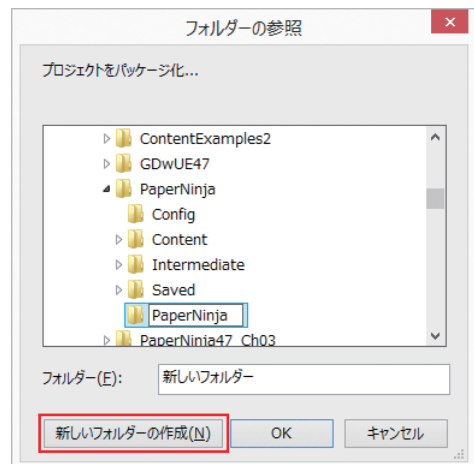
スプラッシュ画像やゲームアイコン画像は **BMPフォーマット** で作成する必要があります。2の倍数サイズでない場合は警告が表示されますが、実害はありません。

エディタスプラッシュを差し替えた方は、一旦ここで未保存のアセットをすべて保存し、PaperNinja プロジェクトのアンリアルエディタを再起動してみましょう。起動時のスプラッシュが独自の絵に置き換わっているはずです。

E.2.3 はじめてのパッケージング(ビルド)

いよいよパッケージングの実作業になります。ここまでの設定は1度やればよいものでしたが、パッケージング作業は何回か行うことになるため、しっかり手順を確認しておきましょう。

1. レベルエディタに戻り、アセットをすべて保存します。
2. メニューバーの[ファイル]>[プロジェクトをパッケージ化]>[Windows]>[Windows (64ビット)]を選択します。
3. パッケージを書き出すフォルダを指定するためのダイアログが開きます。[新しいフォルダーの作成]を選択し、任意のパスに「PaperNinja」という名前のフォルダを作成します(図 E.16)。フォルダを作成できたら、このフォルダを選択して[OK]ボタンをクリックします。一旦フォルダを確定させると、以降は自動的にセットされますので、2回目以降はこのダイアログが開いたら原則 [OK] ボタンを押すだけで済みます。



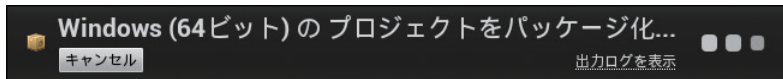
図E.16 パッケージを書き出すフォルダを指定する

Tip

出力フォルダの変更

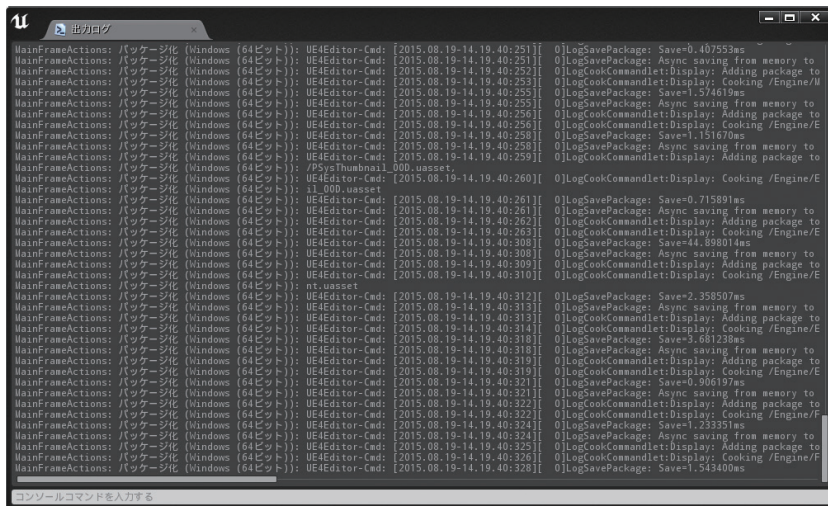
このダイアログ以外の変更場所としては、[プロジェクト設定] ウィンドウの左側で[プロジェクト]>[パッケージ]を選び、[Project]>[Staging Directory]の[...] ボタンをクリックすることで、出力するフォルダを変更することができます。この場合、プロジェクトの共通設定(チーム共有設定)になりますので、その点に注意しましょう。

図 E.17 のようなメッセージがデスクトップの右下に表示され、パッケージ化が開始されます。パッケージ化は処理に時間がかかるため、バックグラウンドで行われます。



図E.17 パッケージング中の表示

図 E.17 内の [出力ログを表示] をクリックすると、図 E.18 のようなログが表示されます。万が一パッケージ化がうまくいかない場合は、ログに書かれている情報を元に原因を探ることが可能です。また、パッケージ化をキャンセルしたい場合は、「キャンセル」をクリックすることでパッケージ化を途中で止めることができます。



図E.18 出力ログ

4. パッケージングが完了するまで(「パッケージングが完成しました」というメッセージが表示されるまで)待ちます(図 E.19)。



図E.19 パッケージ化完了のメッセージ

E.2.4 パッケージ化されたゲームを起動する

パッケージ化したゲームを下記の手順で起動し、内容を確認しましょう。

1. 出力フォルダの「WindowsNoEditor」フォルダに「プロジェクト名.exe」という名前で、実行ファイルが生成されます(例「PaperNinja.exe」)。このexe ファイルをダブルクリックして、ゲームを起動します。
2. タイトル画面で[Game Start]ボタンをクリックし、ステージ1が遊べることを確認しましょう。
3. [＠]キー(英語キーボードの場合は[`]キー)を押してコンソールを開き、「fullscreen」と入力して、ゲーム画面をフルスクリーンにして遊びます。
4. 遊び終わったら、もう一度コンソールを開いて「quit」と入力し、ゲームを終了します。

Tip

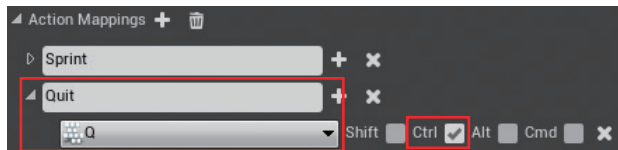
配布するときは...

友達に配布するときなどは、「WindowsNoEditor」フォルダ以下を丸ごと渡すことになります。もし、製品としてパッケージングを行う場合は、通常のWindows 実行ファイル配布と同様に、インストーラを用意するなど、諸々の準備を進める必要があります。

E.2.5 Shippingビルドとビルドのイテレーション

デフォルトでは、パッケージ化(ビルド)は「開発(Development)」設定で行われるため、コンソールコマンドを使用することができます。 負荷調整や不具合調査のためにビルドする場合は、各種のデバッグコマンドが使用可能なDevelopmentビルドを行うべきでしょう。しかし、最終製品版のビルドを行うなら、Shipping設定の方がより適切です。これは制作したゲームを最終的なユーザーへ届ける際に行うビルドで、さまざまなデバッグ関係の機能がオミットされ、コンソールコマンドも使用できなくなります。

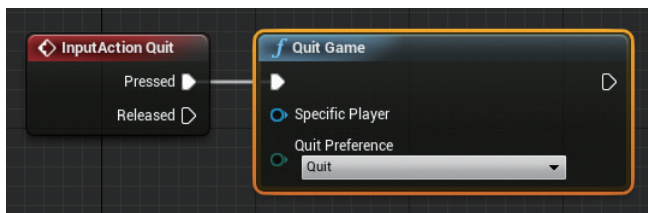
1. メニューバーの [ファイル] > [プロジェクトをパッケージ化] > [ビルドコンフィグレーション] > [出荷(Shipping)]を選択して、ビルド設定を「開発」から「出荷」に切り替えます。
2. ゲームが終了できなくなってしまうので、ツールバーから[設定]>[プロジェクト設定]を選択し、[入力]のメニューを開いて、図E.20のような設定のアクション「Quit」を追加します。



図E.20 新しいアクション「Quit」の追加

3. [コンテンツ>Blueprints] フォルダの [PNPlayerController] のブループリントエディタを開き、[EventGraph]タブをクリックして、グラフ上の空きスペースに[入力]>[アクションイベント]>[Quit]を選択します。

4. [Quit]ノードの[Pressed]実行出力ピンからワイヤーを伸ばし、[Game]>[Quit Game]を選択します(図E.21)。「コンパイル」と「保存」も忘れずに行っておきます。



図E.21 ゲーム終了の操作

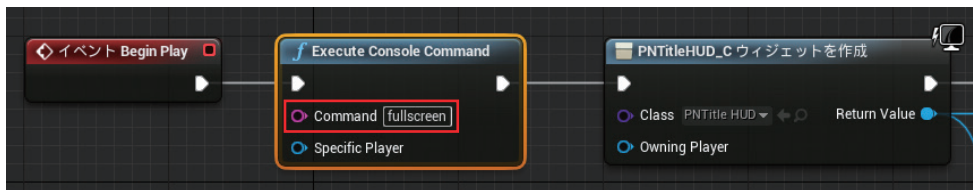
5. [ファイル]>[プロジェクトをパッケージ化]>[Windows]>[Windows (32ビット)]を選択し、出荷設定のパッケージングを開始します。UE4.7の時点では、“開発”時は64ビット、“出荷”時は32ビットしか選択ができません。
6. パッケージングが終わったら、出力された.exeファイルをダブルクリックしてゲームを起動します。[@]キー(もしくは[`]キー)を押してもコンソールが開かないことを確認した上でステージ1へ進み、[Ctrl]+[Q]キーでゲームが終了することを確認しましょう。

Tip

タイトルでも終わらせるには

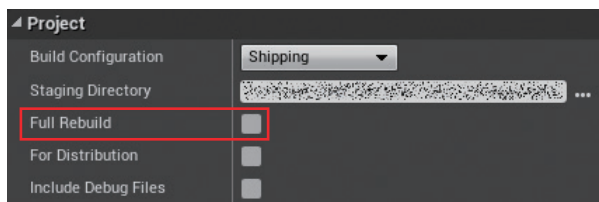
タイトル画面のレベル[PNTileMap]のゲームモードは[PNTileMode]であり、このモード専用のプレイヤーコントローラ[PNTilePlayerController]は前述のQuit操作をサポートしていません。手順4のノードネットワークを[PNTilePlayerController]上で構築すれば、タイトル画面でも終了ができるようになります。

7. タイトル画面で自動的にフルスクリーンがかかるように、E.2.4節では手入力したコンソールコマンド「fullscreen」を、ブループリントのスクリプト上で呼び出すようにしましょう。[コンテンツ>Maps]フォルダの[PNTileMap]をダブルクリックして開き、さらに、レベルブループリントエディタを開きます。
8. [BeginPlay]ノードの実行出力ピンからワイヤーを伸ばし、[Development]>[Execute Console Command]を選択して、[Command]入力ピンに「fullscreen」と入力します(図E.22)。



図E.22 コンソールコマンド呼び出しの埋め込み

9. パッケージングの繰り返しを高速化するために、メニューバーで[ファイル]>[プロジェクトをパッケージ化]>[パッケージング設定...]を選択してウィンドウを開き、[Project]>[Full Rebuild]のチェックを外します(図E.23)。これにより、以降のパッケージングは差分のみが行われるようになります。



図E.23 [Full Rebuild]

10. プロジェクト設定ウィンドウを閉じて、[ファイル] > [プロジェクトをパッケージ化] > [Windows] > [Windows (32ビット)]を選択し、実行ファイルをビルドします。

Shippingビルドであっても、ゲームの起動方法はDevelopmentビルドと変わりません。ゲームを起動して、不具合がないことを確認しておきましょう。

Tip

モバイルへのパッケージング

モバイルでUE4のプロジェクトを動作させる場合にはいくつかの注意点があります。UE4のバージョンアップに伴い注意点も変化していきますので、下記のウェブページで最新の情報を確認し、モバイルでの制限などをチェックしてください。

- モバイルゲームの開発：

<https://docs.unrealengine.com/latest/JPN/Platforms/Mobile/index.html>

また、Androidに特化した情報(SDKのインストール等)や、iOSに特化した情報については、それぞれ下記のページを確認し、必要なセッティングを行ってください。

- Android：

<https://docs.unrealengine.com/latest/JPN/Platforms/Android/index.html>

- iOS：

<https://docs.unrealengine.com/latest/JPN/Platforms/iOS/index.html>

E.3 最後に

E.3.1 本書完全読了後の情報源

本書読了時点で、読者の方の手元には、それぞれに組み上がった「ペーパーニンジャ」ゲームがあるはずですが、このサンプルゲームを軸にして手軽に吸収できるノウハウが、まだもう少しだけ残っています。筆者の個人ブログに誘導する形で誠に恐縮ですが、下記URLでは「Unreal Engine 4で極めるゲーム開発 非公式ページ」と称して、本書中に組み上げなかったマテリアル等の解説、ページ数の関係でカットした小記事など、若干の情報を追記していく予定です。ブックマークの上、気が向いたときにご訪問ください。

- Unreal Engine 4で極めるゲーム開発 非公式ページ：

<http://minahito.hatenablog.com/entry/GameDevelopmentWithUE4>

また、国内の（日本語でやりとりできる）情報源をいくつかリストします。下記 URL のほか、ツイッターでは「#UE4」タグや、「#UE4Study」タグを使って多くの人が情報交換をしていますので、ツイッターユーザーの方はチェックしておくといでしょう。

- UE4 公式ブログ (日本語):
<https://www.unrealengine.com/ja>
- UE4 公式ツイッター (日本語):
<https://twitter.com/UnrealEngineJP>
- 株式会社ヒストリア UE4 ブログ:
<http://historia.co.jp/archives/category/ue4>

E.3.2 次は何をすればいい?

付録を含めて本書を完全に読み終わったあとで、読者の方々がどこに向かって進んでいくのか、筆者には想像することしかできません。

- 読書前から作りたいゲームがあり（あるいは読書中に作りたいゲームが思い浮かんで）、すぐに新しいゲームプロジェクトに取り掛かりたい人。
- 本の内容は完全に理解できたが、作りたいゲームがまだ固まっていない人。
- 言われるがままにハンズオンしたが、理解できたか不安で、もう一度読み返そうと思っている人。

等々、それぞれでしょう。いずれの道を行くにせよ、長い時間をかけて本書を読み込んできたわけですから、その内容が血肉になっているかどうかが重要です。本書としては、復習として本書を読み返すより、ここまでの作業結果をベースにして、ステージ2、ステージ3……と**新しいステージをゲームに追加していくこと**をお勧めします。これがもっとも適切な復習になります。特に、チームでラーニングをしているのであれば、チーム内の役割分担を守ったまま、第4章で解説したワークフローで作業を進めてみましょう。

- ステージ2はアセットやブループリントを新作したり、改変したりはせず、本編の第29章終了時点の材料だけで作ってみましょう。本書の復習はこれで十分です。
- ステージ3はAppendix A～Bで追加したアセット等、本編で扱っていないギミック（歯車や、各種プロジェクトイル、落下する床など）も組み込んで作ってみましょう。これらのアセットはまだ調整の余地があるため、ちょっとした応用を求められ、何度も本編や付録の各解説文を参照することになるでしょう。
- ステージ4では、読者の（あるいは、この追加ステージ制作に参加しているメンバーの）専門職に合わせてアセットを新作し、組み込んだもので作ってみましょう。アーティストがいるなら、新しい建築材、プロップ、キャラを、アニメーターがいるなら新しいアニメーションを、レベルデザイナーは新しいギミックを作り、エンジニアは異なる意思決定を行うビヘイビアツリーを持った敵を作ってみてください。

この3ステップは、短いながらも「守破離」の考え方に近いお勧めのステップです。短くて2日、長くても5日あれば、ステージ4まで作れるはずです。**ここでやり方が分からず引っかかることは、どのみち本番でも引っかかるのですが、確実に動作するプロジェクトで七転八起するのと、本番で七転八起するのでは、どちらがダメージが軽く、時間が短く済み、チームなりの対策を持てるかは明白ではないでしょうか。**筆者としては、確実に動作するプロジェクトを使ってコンパクトにやることをお勧めします。

■ Note

MODで学ぶ

独自のゲームを生み出すことが目的の私たちにとって、既存のゲームの拡張や改造は興味の対象外かもしれませんが、この手の取り組みは、求められる知識や技術がプロダクションのそれに近いため、抜群のラーニング効果があります。このような拡張や改造をMOD (Modification) といい、PCゲームの世界では、伝統的な楽しみ方のひとつであり、海外のゲーム産業においては、人材輩出の分厚い基盤にもなっています。特にゲームエンジンを使って作ったタイトルでは、ゲームエディタを製品版ゲームに同梱して出荷することが容易なため、非常に手軽にMODに取り組むことができ、Epic Gamesのタイトル『アンリアルトーナメント』はもちろん、UE4で作られたオープンワールドのゲーム『ARK: Survival Evolved』もMOD対応を謳っています。ペーパーニンジャがイマイチという人は、このようなゲームを入手してラーニングにあててみるのも悪くありません。

E.3.3 困ったら誰に聞けばいい?

サポート料を払ってUE4を使用しているチームは、「公式サポート」を利用できますが、無料で使用している場合はコミュニティベースのサポートシステムが頼みになります。

- AnswerHub 日本語トップ：
<https://answers.unrealengine.com/spaces/16/japanese.html>
- Facebook Unreal Engine ユーザー助け合い所：
<https://www.facebook.com/groups/unrealuserj>

困ったときに掲示板やAnswerHubへ助けを求めるときは、有料サポートではない点を頭に入れておく必要があります。まずは検索機能を使って、同様の問題を抱えている人がいないかどうか、またその問題が既に解決されていないかどうかを必ずチェックしましょう。次に、問題を投稿する際は、できるだけ回答を得やすくするために、多少手間でも情報を初めに揃えておくことが大切です。UE4の環境(OSやバージョン、GPUの情報など)、トラブルのスクリーンショット(動画があればベター)、スクリプトもスクリーンショットを貼るようにしましょう。そのほか、「自分が問題を投稿したあとは、必ず〇〇について追加情報を求められているなあ」と感じるがあれば、その〇〇も自分の中のテンプレートに書き加えておきます。中には「今回はこの情報は必要ない」と自分で判断できるものもあるかもしれませんが、回答する側からは問題を投稿している人の知識／技術レベル、バックボーンなどは判断がつきませんので、情報を絞るか開示するかで迷ったら、必ず後者を取るようにしましょう。

■ Column

有料サポートでも有効な詳細報告

これはEpic Gamesの有料サポートについて述べているコラムではなく、一般化した話であり、筆者の肌感覚を述べているだけです。その前提で読んで欲しいのですが、前述のような詳細な情報提供はミドルウェアやゲームエンジンのサポートにおいて有効な回答を短期間で得るための共通のコツのような気がしています。同じ緊急レベルのサポートチケットが並んでいるときには、情報が詳細で解決しやすいものから先にサポートが実行されていくのが人情ではないか という程度の根拠ですが、そのように感じられることが実際に少なくありません。

ただし、より詳細な情報提供ということになるとどうしてもエンジニアがレポーターを務めることになり、エンジニアの馬力数が少ないプロジェクトでは、レポートや、やり取りの代行もそれなりの負荷となります。難しいことですが、少しずつレポートのテンプレートを作っていく、当事者が直接レポートを投稿できるようになっていくのが理想です。もちろん、エンジニアの人数が多いプロジェクトでは、トラブルシューティングをエンジニアに一手に集めて、他の職種からその負担を取り除くということも選択肢のひとつになります。

また、解決したときは、回答がどの要素が有効であったか、自己解決した場合はその解決手段について追記しておきます。これは、将来同様のトラブルを抱えた人への回答になりますし、バージョンアップ修正の手がかりにもなりますので、非常に重要です。

一歩進んで、「他の人が困っているときに回答を寄せる」こともかなり勉強になりますし、ギブ&テイクの理念にも沿ってますので、お勧めします。実際に回答にチャレンジしてみると、投稿時の情報がどれほど重要か理解できるようになり、自分が問題を投稿するときもポイントを押さえやすくなるはずです。言うまでもなく、これらのことはUE4に限らず、すべてのコミュニティベースの助け合いにおいて、とても大切なことです。

Column

QAのお手本

ゲーム会社に勤務している人であれば、開発中盤～終盤にかけて QA（品質保証）が動き出すため、プロのデバグが書いたレポートを読むチャンスがあるはずです。QAやデバグは大量の不具合レポートを扱う関係上、1回のレポートで必要な情報をすべて伝達するノウハウを持っており、かなりのお手本になります。QAはエンジニア事案だと思っている方も、良い機会と考えて、デバグのレポートを読んでもみるとよいでしょう。

E.3.4 まだやることがあるってこと？

本編と付録を合わせて 750 ページ近頃の旅路であったにも関わらず、「ペーパーニンジャ」の仕様にありながらカバーできなかった要素と、UE4 の入門本としてカバーしきれなかった要素があります。

複数スイッチとイベント(レベルデザイナー向け)

p.53 に「何個かスイッチを踏まないと開かないフスマがある」といったゲームの紹介文がありますが、本編と付録、そしてステージ 1 を通じて、このギミックを扱うチャンスがありませんでした。ちょっとしたフロー制御ですので、前述の書籍非公式ページで解説予定です。

セーブとロード(エンジニア向け)

ペーパーニンジャの紹介文 (p.55) にありながら、本編と付録を通じて、「集めたコインを貯金して、先のステージをアンロックする」仕組みを実現する余力がありませんでした。セーブするかどうかはともかく、レベルをつないでゲームを作る構成上、複数レベルをまたいで持つ情報の持ち方はすぐに欲しい情報かと思います。こちらについては「GameInstance」をキーワードにインターネット上で情報収集をお願いします。

ブループリントデバグ(全職種)

ビジュアルスクリプト言語はデバグも強力で、筆者は UE3 の「Kismet」や、UE4 の「ブループリント」で、いわゆる「非技術者」にグループされる開発者が、自分たちで問題を次々に解決するところを見てきました。初級教本を担うと宣言したこの本では、すべての方にデバグのテクニックを伝える役目を果たすべきだったのですが、カバーしきれませんでした。こちらに関しては公式ドキュメント(下記 URL)をご参照ください。

<https://docs-origin.unrealengine.com/latest/JPN/Engine/Blueprints/UserGuide/Debugging/index.html>

バージョンコントロール(全職種)

個人制作であれ、チーム制作であれ、業務であれ、非業務であれ、必ずバージョンコントロールシステムを使うはずですが、環境セットアップの関係で扱うことができませんでした。幸い公式にドキュメントがありますので、下記 URL をご参照ください。

- セットアップの情報(エンジニア向け):
<https://docs.unrealengine.com/latest/JPN/Engine/Basics/SourceControl/Perforce/index.html>
- コンテンツブラウザ上の操作(全職種向け):
<https://docs.unrealengine.com/latest/JPN/Engine/UI/SourceControl/index.html>

C++ (エンジニア向け)

本書はゲーム中のすべての処理をブループリントスクリプトで実現しており、主にエンジニアが使用することになる「C++」を一切取り上げていません。本来であれば(本編中に何度か述べたように)、チーム内のエンジニアとそれ以外の職種のバランスをみて、適度にC++を取り入れたほうが、チーム全体の生産性は確実に上がります。エンジニアしか編集しない機能(単体機能)はエンジニアがテキストのプログラミング言語で作り、デザイナーがそれを組み合わせるという作り方の気持ちよさは、UE3の頃から立証されていることですが、本書の主旨上、扱うことができませんでした。

例えば、プレイヤーキャラや敵の移動は、標準の移動コンポーネントに任せている関係上、切り返しや、ターンの動きがアニメーションとうまくシンクロしていません。このあたりはC++でコードを書いたほうが作り込めるのですが、いつか解説のチャンスがあれば……とも思いますが、一旦各自の課題とさせていただきます。

マチネ(レベルデザイナー、アニメーター向け)

カットシーンエディタの「マチネ」は、本書上で何度か名前が出ていながら(しかも、「ペーパーニンジャ」のようなプラットフォームでは強力なレベルデザインツールになることが分かっているが)、取り上げませんでした。UE4のマチネは、UE3よりいくらかパワーダウンしており、新しいシーケンスエディタに移行する時期が近づいていることもあって、優先度を下げたものです。

■ Tip

「ナイアガラ」がリリースされたら...

余談ながら、UE3時代から使われてきた現行パーティクルシステムエディタ「カスケード」も、ノードベースの新エディタ「ナイアガラ」に変更されることが予告されていますが、こちらはパーティクルシステム入門の要点はエディタに左右されるものではないことから、書き方を工夫して第27章に収録しました。ナイアガラがリリースされたら、同じソースアセットを使って同じ動きのパーティクルシステムを作れば、変更要点を速やかに把握できるはずです。

マルチプレイヤー(エンジニア向け)

本編や付録で学んだ「ボーンとコントローラの分離」「コントローラとカメラ」の設計関係は、腑に落ちる人もいれば、腑に落ちない人もいたことでしょう。これらの仕組みは、シングルプレーでも効果を発揮しますが、特にマルチプレイヤーゲームパートを制作したときに活きることになります。

VR (エンジニア向け)

『ペーパーニンジャ』はご存知の通り(?)リビングに置いてある紙細工のオモチャが動き出すというコンセプトのゲームにする予定でした。VRを装着した場合は、右サムスティックでカメラを回転させる機能をオミットして、プレイヤーが実際にこのミニチュアの周りを動いて遊ぶという体験に仕上げたかったのですが、またの機会としましょう。VRについては、下記URLをご参照ください。

- Oculus Rift クイックスタートガイド：
<https://docs.unrealengine.com/latest/JPN/Platforms/Oculus/QuickStart/index.html>
- Samsung Gear VR 情報のトップページ：
<https://docs.unrealengine.com/latest/JPN/Platforms/GearVR/index.html>

ランドスケープとフォリッジ(環境アーティスト向け)

『Skyrim』のようなゲームを作りたい人にとっては、本書で取り上げたブラシ機能より、「ランドスケープ」と「フォリッジ」のほうが重要かと思います。こちらについては公式マニュアルをご覧ください。

- ランドスケープのクイックスタートガイド：
<https://docs.unrealengine.com/latest/JPN/Engine/Landscape/QuickStart/index.html>
- フオリッジ情報のトップページ：
https://docs.unrealengine.com/latest/JPN/Resources/ContentExamples/Landscapes/1_3/index.html

ただし、機能については公式マニュアルで学ぶことができますが、ワークフロー（オープンワールドにおけるレベルエディット）は世界中のゲーム会社で試行錯誤されており、確立されたやり方はまだありません。オープンワールドが現実に近いほど、そして、密度が高ければ高いほど、「レベルデザイナーがグレーボックスを作り、アーティストがグレーボックスをテンプレートにしてメッシングを行う」という段階工程が組みづらくなります。当面は、チーム内で定期的な会合を設けて、手戻りや作業上の混乱について話し合って対策を練るほか、GDCやCEDECといった最新のゲーム開発者向けカンファレンスで、各社の試行錯誤の情報を集めていくことになるでしょう。

謝辞

本編最終章にクレジットさせていただいた制作協力スタッフの皆様、Appendix D の 2D 素材を制作してくださいました 2D ファンタジスタ 吉川愛子様、度重なる不切変更や当初のページ数見込みを大幅にオーバーしながらも全情報を収録してくださった (株) ボーンデジタル様と編集の堀越祐樹様に深い感謝を申し上げます。所属先である (株) バンダイナムコスタジオからは有形無形の強力なサポートをいただきました。また、日本では (株) ソフトバンククリエイティブから出版されております「Unityではじめるゲームづくり(原題: Game Development with Unity)」の素晴らしい構成を、本書構想時点から一貫して手本とさせていただきます。著者のミシェル・メナード女史に、この場を借りて感謝申し上げます。