

じゃんけんゲーム

よりゲームらしくするための基本機能を学習する

v1.0

Index

1. 作成するゲーム内容、得られる知識
2. プロジェクトの作成
3. じゃんけんゲームの作成
4. 列挙型を使ってエラーが少ない処理にする
5. 配列を使って5回勝負の結果を出力する
6. 構造体を使って出力する結果の情報を増やす
7. じゃんけんゲームのUI作成
8. GameMode,GameStateを作成して、ゲームの状態を中継する
9. ウィジェットブループリントのイベントをブループリントで受け取って処理する
10. 結果を表示するUI作成
11. タイトルを表示するUI作成
12. メニュー画面の作成
13. 音の追加
14. 解像度設定
15. テスト項目票の作成・デバッグ
16. パッケージ化

1. 作成するゲーム内容、得られる知識
2. プロジェクトの作成
3. ジャンケンゲームの作成
 - 3.1 新規レベル (Game) を作成・保存
 - 3.2 ブループリント(BP_JankenGameController)を作成
 - 3.3 簡単なジャンケンゲームを実装
 - 3.3.1 プレイヤーはキーボード入力でグー、チョキ、パーを出力
 - 3.3.2 関数:PlayEnemyJanken (敵はランダムでグー、チョキ、パー)
 - 3.3.3 プレイヤーと敵のジャンケン結果から勝敗を判定する関数:JudgeJankenを作成
 - 3.3.4 作成した関数を使ってジャンケンゲームを実装
 4. 列挙型を使ってエラーが少ない処理にする
 - 4.1 列挙型:EJankenを作成
 - 4.2 列挙型:EJankenResultを作成
 - 4.3 関数:PlayEnemyJankenのアウトプットの型をEJankenに変更
 - 4.4 関数: JudgeJankenのインプットの型をEjanken, アウトプットの型をEJankenResultに変更
 - 4.5 関数のインプット・アウトプットの変更によるエラー修正
 5. 配列を使って5回勝負の結果を出力する
 - 5.1 フローを5回勝負に変更する
 - 5.2 変数の追加(配列の変数)
 - 5.3 配列の操作
 - 5.4 フローの実装
 - 5.5 カスタムイベントを作成して処理を切り離す
 6. 構造体を使って出力する結果の情報を増やす
 - 6.1 構造体:FJankenResultを作成する
 - 6.2 変数の追加(構造体の配列の変数)
 - 6.3 bWinResultsを使用している箇所をFJankenResultsに置き換える
 7. ジャンケンゲームのUI作成
 - 7.1 UIに使用する画像のインポート・設定
 - 7.2 ウィジェットブループリントの操作のおさらい
 - 7.3 ゲームのジャンケン選択画面を作成する
 8. GameMode, GameStateを作成して、ゲームの状態を中継する
 - 8.1 ウィジェットブループリントとゲームロジックの状態を中継する仕組みを作成する
 - 8.2 BP_GameStateを作成する
 - 8.3 BP_GameModeを作成する
 - 8.4 WBP_Gameのウィジェットにバインドを作成して、BP_GameStateの値を反映させる
 - 8.5 WBP_Gameを表示する
 - 8.6 ゲームの状態を中継する仕組みの説明
 - 8.7 BP_JankenGameControllerの変数をBP_GameStateに移行する
 - 8.8 BP_GameStateに関数Initializeを作成し、初期化処理を置き換える
 - 8.9 BP_JankenGameControllerの処理をBP_GameStateの変数を使用するように変更する
 - 8.10 BP_GameStateのCallJankenStrに文字を設定する
 - 8.11 BP_GameStateの変数をWBP_Gameのウィジェットにバインドする

9. ウィジェットブループリントのイベントをブループリントで受け取って処理する

9.1 WBP_GameのButton_Janken_Guをクリックしたら、ゲームを選択した処理を実行するように実装する

9.2 Button_Janken_Choki, Button_Janken_Paも同様にクリックしたら、処理を実行するように実装する

9.3 MouseOverした際にプレイヤーのじゃんけん画像が変更するように実装する

10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

10.6.4 WBP_JankenResultに画像を設定する関数:GetResultGameTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：
SetJankenResultsを作成

10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

11. タイトルを表示するUI作成

11.1 新規レベルTitleを作成

11.2 WBP_Titleの作成

11.3 ウィジェットの配置

11.4 ブループリント：BP_TitleControllerを作成

11.5 WBP_Titleにボタンのイベントディスパッチャを追加

11.6 BP_TitleControllerにボタンのイベントをバインド処理を実装

11.7 Loading画面 WBP_Loadingを作成

11.8 レベル遷移前にLoading画面を表示

11.9 WBP_GameResultのタイトルボタンからTitleレベルに遷移

12. メニュー画面の作成

12.1 WBP_Menuの作成

12.2 WBP_Menuを表示

12.3 WBP_Menuのボタンからイベントディスパッチャを呼び出す

12.4 WBP_Menuのボタンイベントをバインドする

12.4.1 再開ボタンのイベントにバインド

12.4.2 タイトルボタンのイベントにバインド

12.4.3 終了ボタンのイベントにバインド

13. 音の追加

13.1 音データのインポート

13.2 BGMがループするようにキューを作成する

13.3 BGMを再生する

13.4 SEを再生する

14. 解像度設定

14.1 列挙型 EResolutionを作成

14.2 ComboBoxにEResolutionのエニュミレータをすべて表示

14.3 解像度を変える

14.4 レベルを跨いで値を保持するためにBP_GameInstanceを作成

14.5 解像度の設定を保存して、ゲーム開始時に読み込む

15. テスト項目票の作成、デバッグ

15.1 テスト項目票を作成して、テストを実施

16. パッケージ化

16.1 配布用パッケージにするためのプロジェクト設定

16.2 Windows(64ビット)でパッケージ化

16.3 パッケージサイズをより小さくする

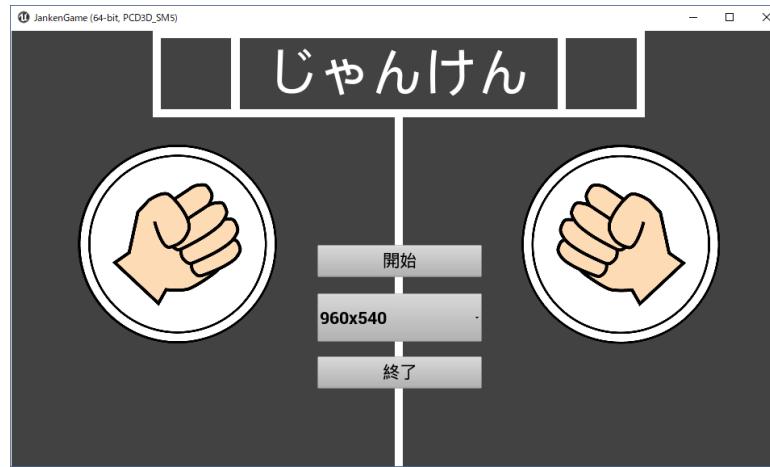
変更履歴

バージョン	日付	変更内容
V1.0	2017/12/05	リリース
V1.1	2019/01/16	画像の修正 4.20.3でも動作を確認

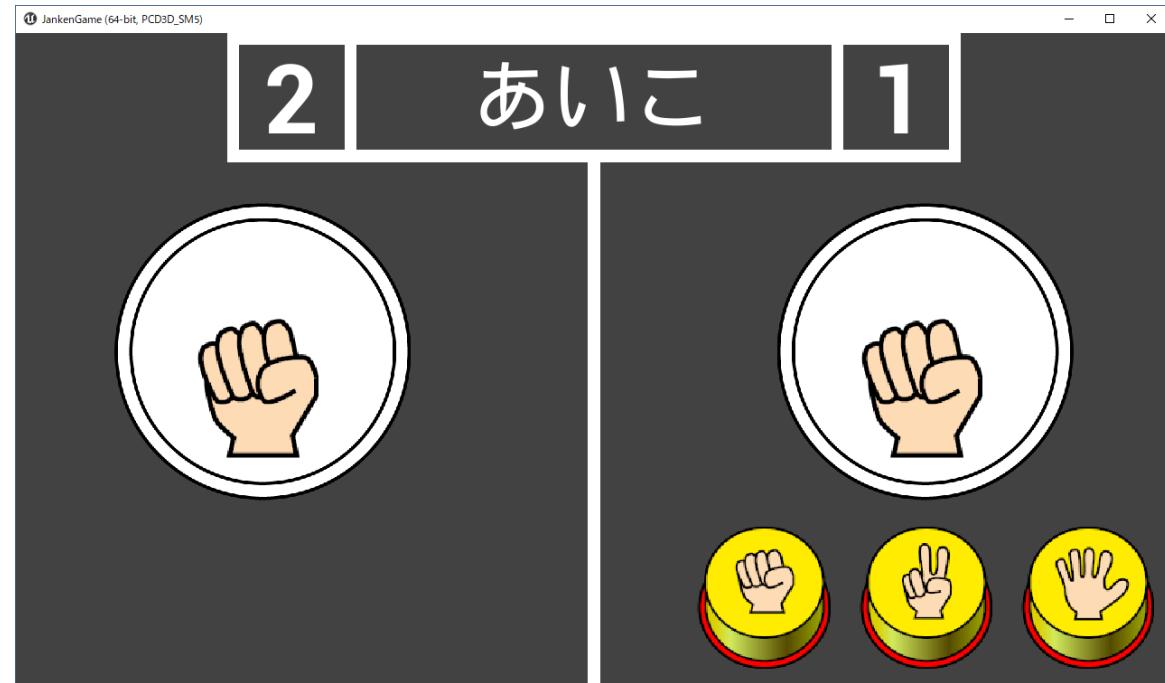
1. 作成するゲーム内容、得られる知識

作成するゲーム内容

タイトル画面



ゲーム画面



メニュー表示

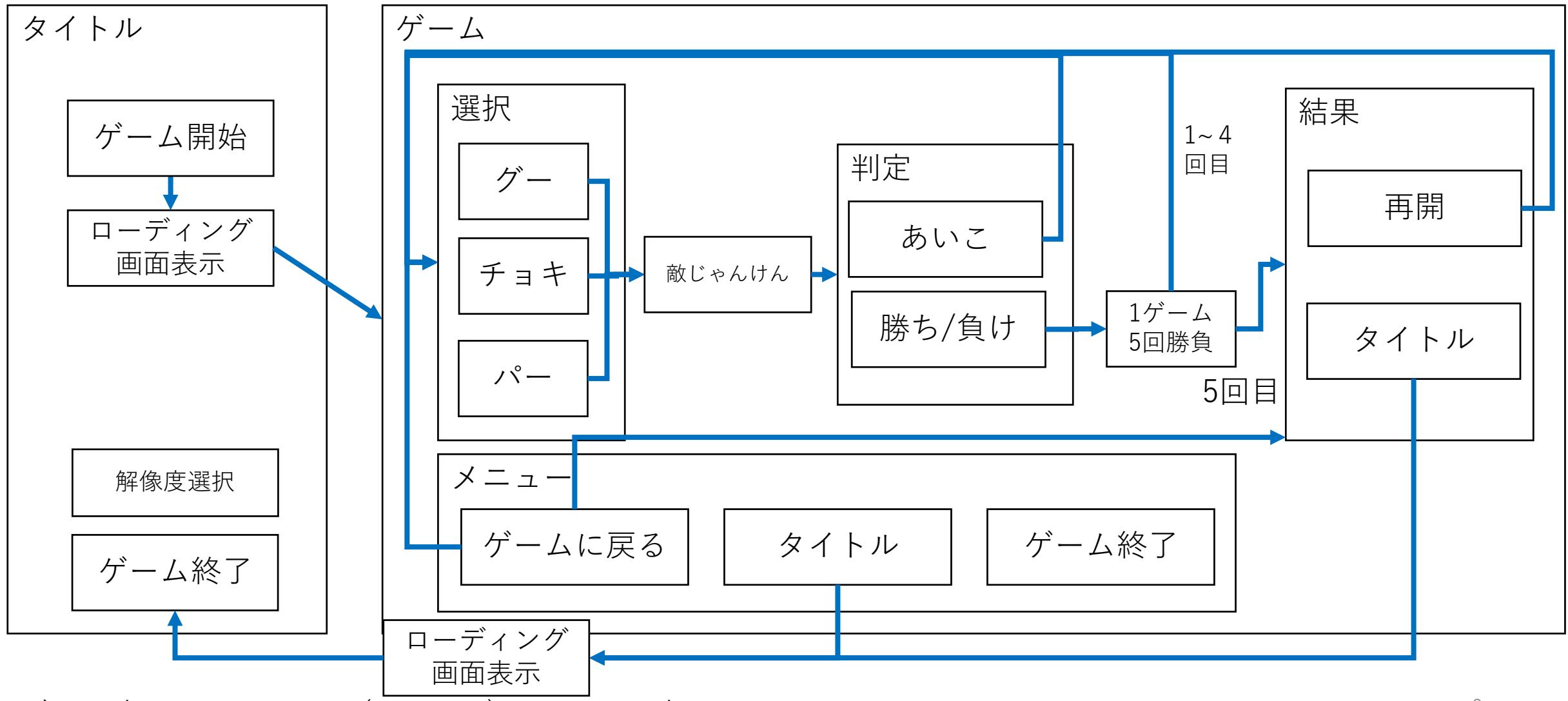


結果表示



最小単位の構成で、レベル間の移動やゲームに必要な要素の組み込み方を学習する

ゲーム構成



得られる知識

- 構造体
- 列挙型
- 配列（作成、取得、クリア）
- 複数レベルの遷移（タイトル、ゲーム）
- GameMode
- GameState
- GameInstance
- ローディング画面
- ボタンのイメージ対応
- ウィジェットの部品化
- メニュー画面
- ゲーム画面のサイズ変更（フルサイズ、解像度指定）
- 設定情報の保存と読み込み

使用するアプリケーション

- UnrealEngine4(Winsows 10) Version 4.18.3
- UnrealEngine4(Winsows 10) Version 4.19.2 動作確認済み
- UnrealEngine4(Winsows 10) Version 4.20.3 動作確認済み

UE4 Style Guideに準拠

The screenshot shows the GitHub repository page for `Allar/ue4-style-guide`. The repository has 87 commits, 2 branches, 0 releases, and 4 contributors. The latest commit was made a day ago. The repository is under the MIT license. The README file contains the following text:

```
Gamemakin UE4 Style Guide() {  
A mostly reasonable approach to Unreal Engine 4  
Heavily inspired by the Airbnb Javascript Style Guide.  
analytics GA lint partial support}
```

ファイル名やフォルダ構成については
UE4 Style Guideに準拠するようにしています

ue4-style-guide

<https://github.com/Allar/ue4-style-guide>

日本語翻訳

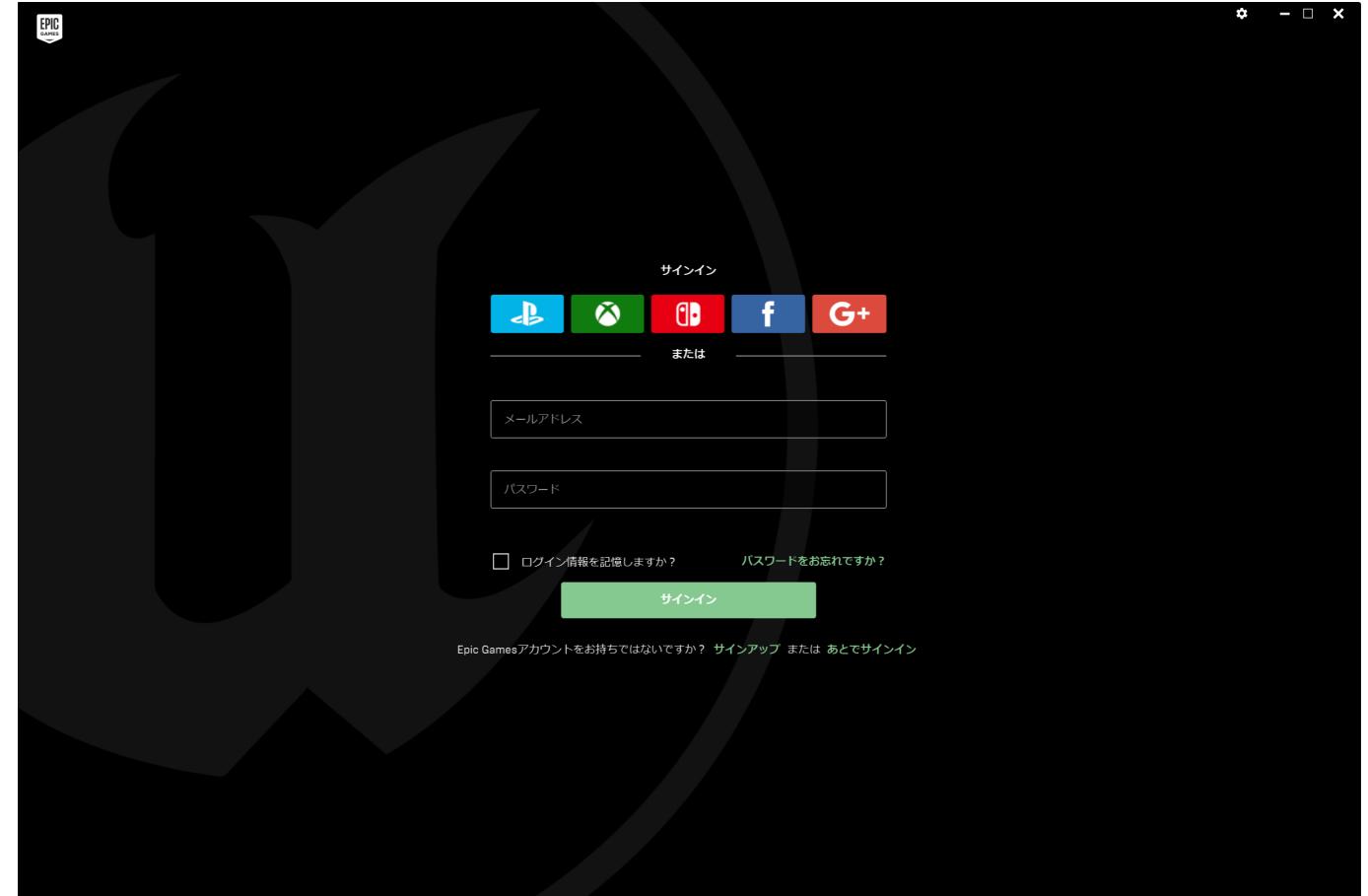
<https://github.com/akenatsu/ue4-style-guide/blob/master/README.jp.md>

UE4 Style GuideとLinterPlugin

http://denshikousakubu.com/2017/11/01/20171101_UE4StyleGuideAndLinterPlugin/

2. プロジェクトを作成する

Epic Games Launcherにログイン



Epic Games Launcherを
立ち上げる

メールアドレス・パスワードを入力してログイン

バージョンを選択して起動

1. UNREAL ENGINEを選択
2. 起動横の▼をクリック
3. 指定のバージョンを選択
[4.19.2]を選択
4. 起動ボタンをクリック



作成するプロジェクトの設定

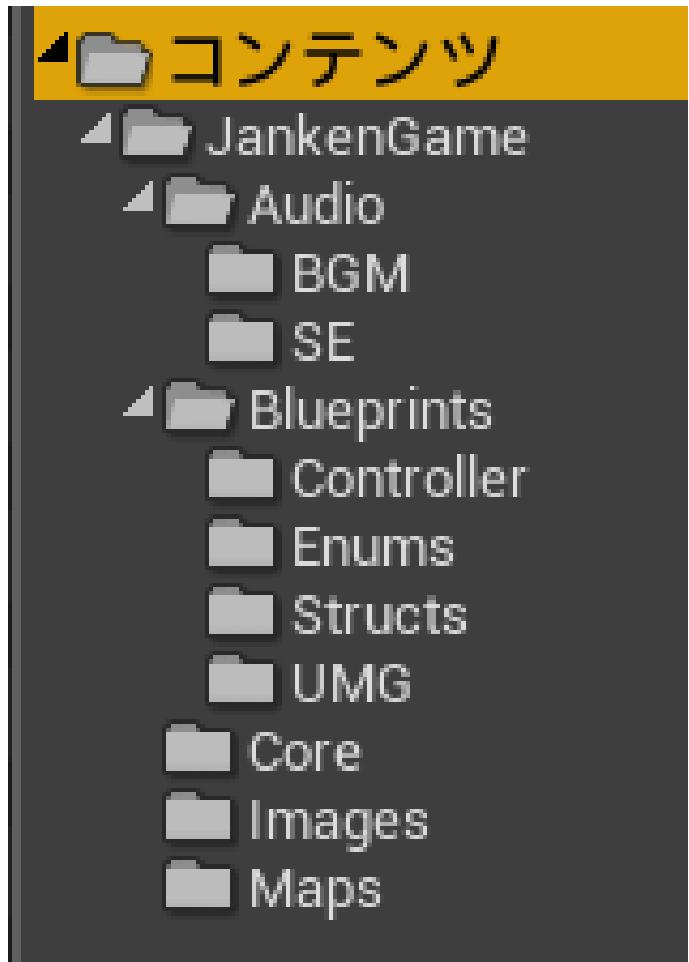
1. 新規プロジェクトを選択
2. ブループリントタブを選択
3. 空のプロジェクトを選択
4. スターターコンテンツ無し
5. フォルダを設定
6. プロジェクト名を設定
プロジェクト名 : JankenGame
7. プロジェクトを作成をクリック



プロジェクトブラウザの詳細

<https://docs.unrealengine.com/latest/JPN/Engine/Basics/Projects/Browser/index.html>

フォルダを作成する



3.じゃんけんゲームの作成

3.じゃんけんゲームの作成

3.1 新規レベル（Game）を作成・保存

3.2 ブループリント(BP_JankenGameController)を作成

3.3 簡単なじゃんけんゲームを実装

3.3.1 プレイヤーはキーボード入力でグー、チョキ、パーを出力

3.3.2 関数:PlayEnemyJanken (敵はランダムでグー、チョキ、パー)

3.3.3 プレイヤーと敵のじゃんけん結果から勝敗を判定する
関数:JudgeJankenを作成

3.3.4 作成した関数を使ってじゃんけんゲームを実装

3.じゃんけんゲームの作成

3.1 新規レベル（Game）を作成・保存

3.2 ブループリント(BP_JankenGameController)を作成

3.3 簡単なじゃんけんゲームを実装

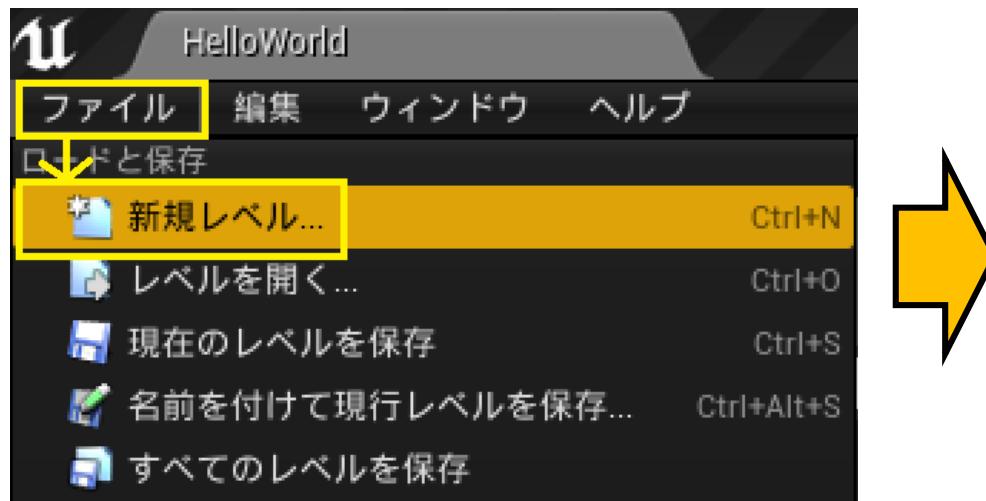
3.3.1 プレイヤーはキーボード入力でグー、チョキ、パーを出力

3.3.2 関数:PlayEnemyJanken (敵はランダムでグー、チョキ、パー)

3.3.3 プレイヤーと敵のじゃんけん結果から勝敗を判定する
関数:JudgeJankenを作成

3.3.4 作成した関数を使ってじゃんけんゲームを実装

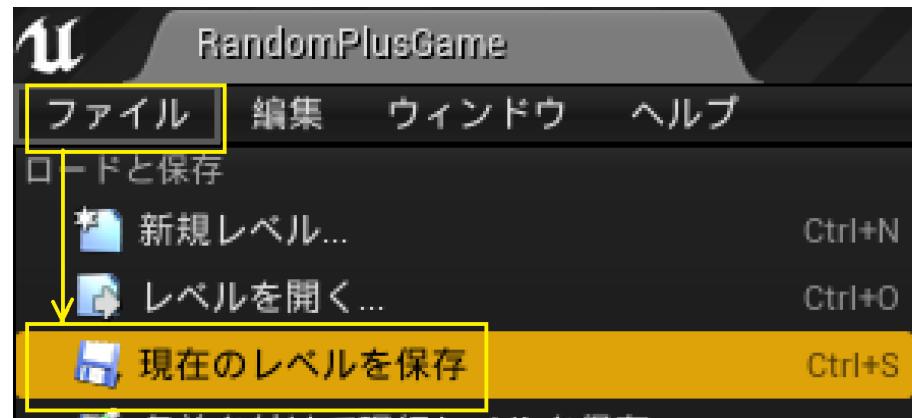
新規レベルの作成



ファイル > 新規レベル
ショートカット : Ctrl+N

[空のレベル]を選択する

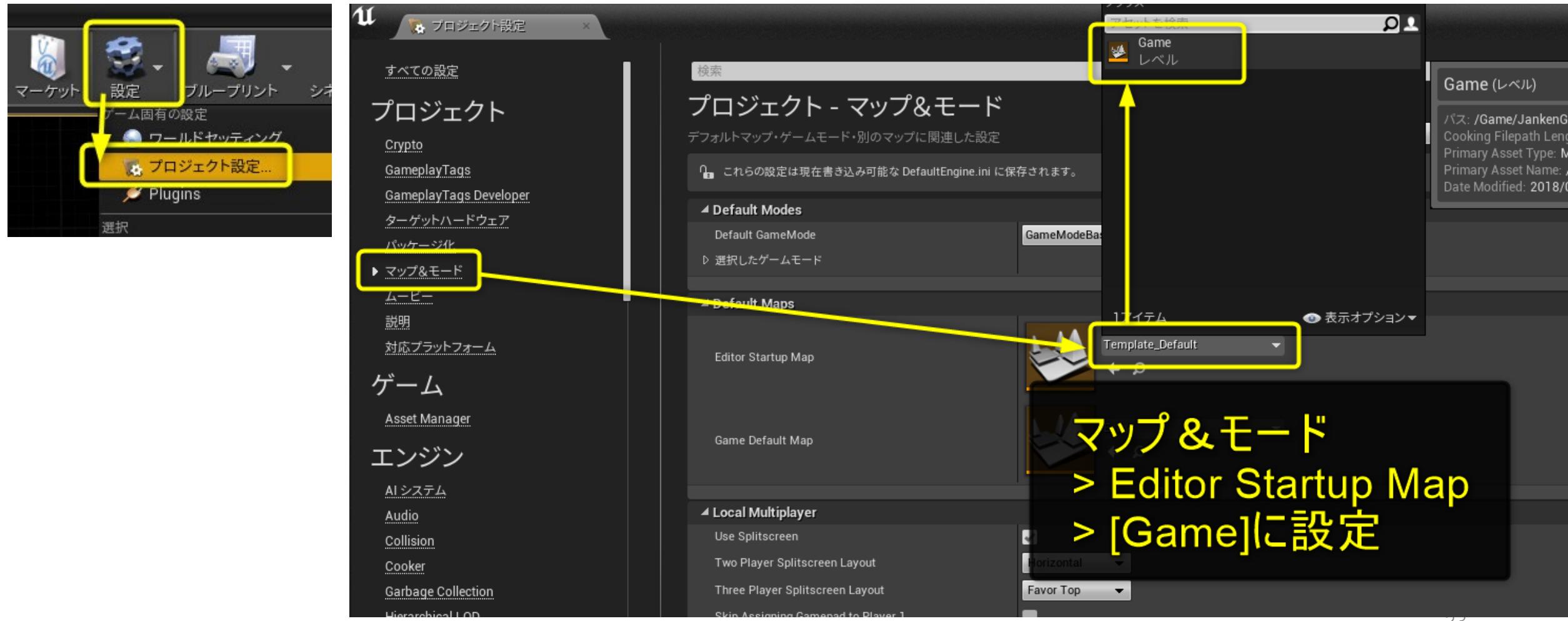
現在のレベルを保存 名前を[Game]に設定して保存



ファイル > 現在のレベルを保存
ショートカット: Ctrl+S

Maps フォルダを選択
> 名前を [Game] に設定
> [保存] をクリック

Editor Startup MapをGameに設定する



3.じゃんけんゲームの作成

3.1 新規レベル（Game）を作成・保存

3.2 ブループリント(BP_JankenGameController)を作成

3.3 簡単なじゃんけんゲームを実装

3.3.1 プレイヤーはキーボード入力でグー、チョキ、パーを出力

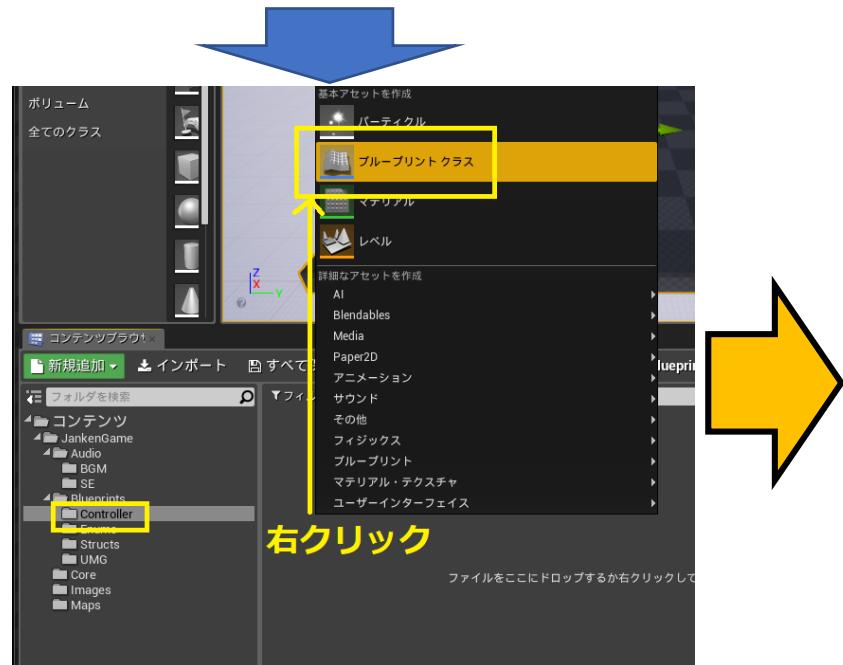
3.3.2 関数:PlayEnemyJanken (敵はランダムでグー、チョキ、パー)

3.3.3 プレイヤーと敵のじゃんけん結果から勝敗を判定する
関数:JudgeJankenを作成

3.3.4 作成した関数を使ってじゃんけんゲームを実装

ブループリント作成の流れ

作成するフォルダ	ブループリント名	親クラス
Controller	BP_JankenGameController	Actor



作成するフォルダを選択
> 新規追加 or 右クリック
> ブループリントクラス



[Actor]ボタンをクリック



名前を設定する
名前を
[BP_JankenGameController]に設定

3.じゃんけんゲームの作成

3.1 新規レベル（Game）を作成・保存

3.2 ブループリント(BP_JankenGameController)を作成

3.3 簡単なじゃんけんゲームを実装

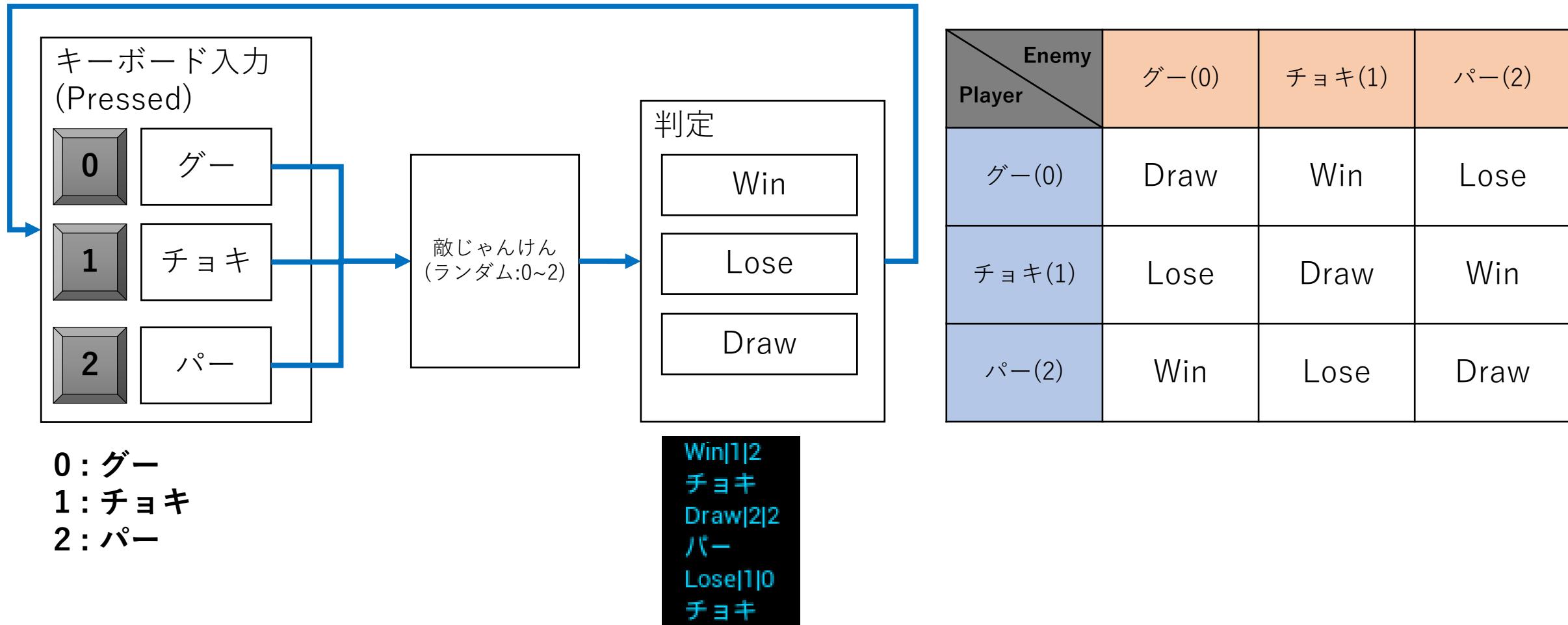
3.3.1 プレイヤーはキーボード入力でグー、チョキ、パーを出力

3.3.2 関数:PlayEnemyJanken (敵はランダムでグー、チョキ、パー)

3.3.3 プレイヤーと敵のじゃんけん結果から勝敗を判定する
関数:JudgeJankenを作成

3.3.4 作成した関数を使ってじゃんけんゲームを実装

3.3 簡単なじゃんけんゲームを実装



3.じゃんけんゲームの作成

3.1 新規レベル（Game）を作成・保存

3.2 ブループリント(BP_JankenGameController)を作成

3.3 簡単なじゃんけんゲームを実装

3.3.1 プレイヤーはキーボード入力でグー、チョキ、パーを出力

3.3.2 関数:PlayEnemyJanken (敵はランダムでグー、チョキ、パー)

3.3.3 プレイヤーと敵のじゃんけん結果から勝敗を判定する
関数:JudgeJankenを作成

3.3.4 作成した関数を使ってじゃんけんゲームを実装

プレイヤーは
キーボード入力で
グー、チョキ、パーを出力



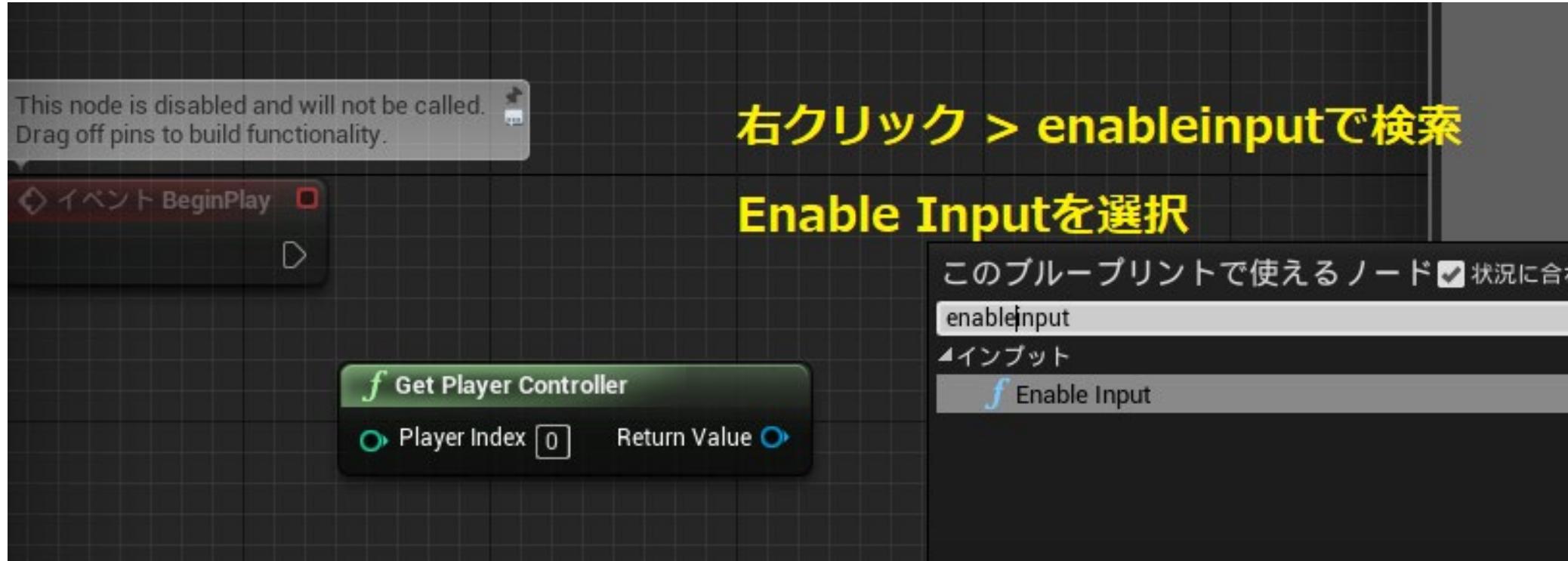
キーボード入力を受け付ける設定 1 Get Player Controllerの追加



右クリック > get player controllerで検索 > Get Player Controllerを選択

キーボード入力を受け付ける設定 2

Enable Inputを追加する



右クリック > enableInputで検索 > Enable Inputを選択

キーボード入力を受け付ける設定3

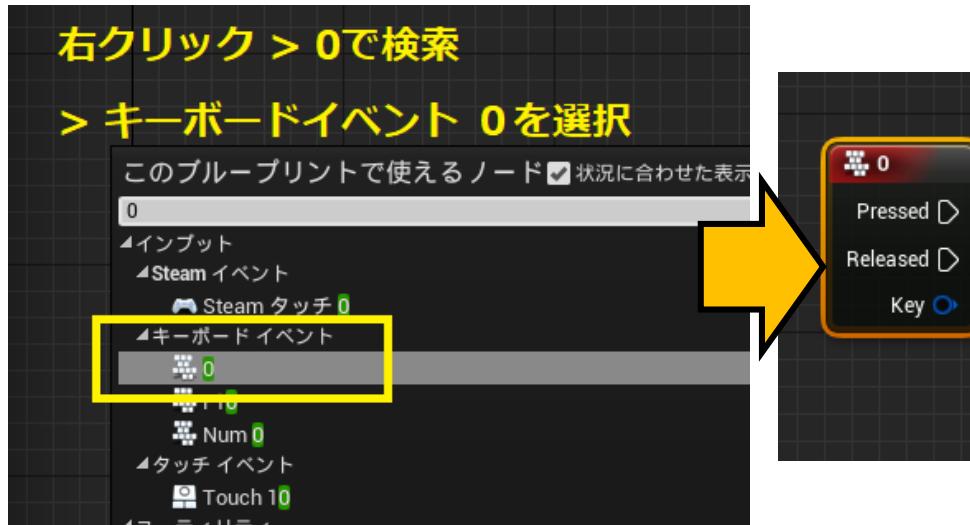


右クリック > enableInputで検索 > Enable Inputを選択

0キーイベントを追加 Print Stringでグーを出力

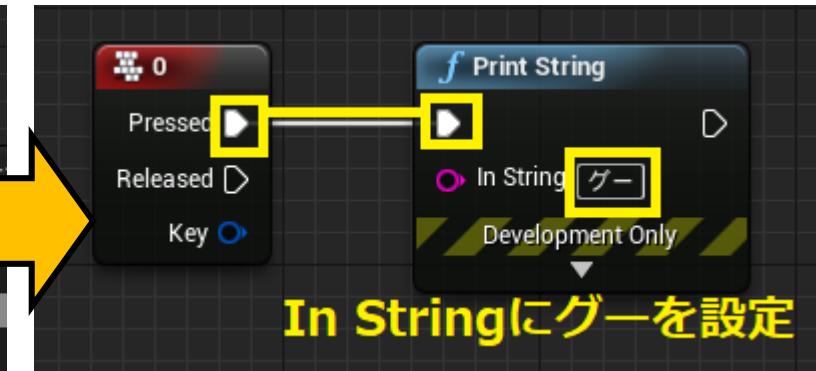
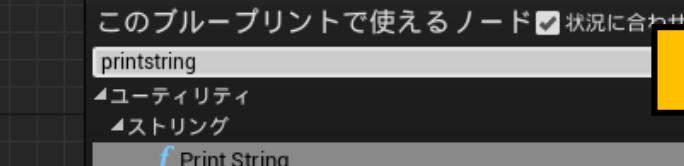
右クリック > 0で検索

> キーボードイベント 0を選択



右クリック > printstringで検索

> Print Stringを選択

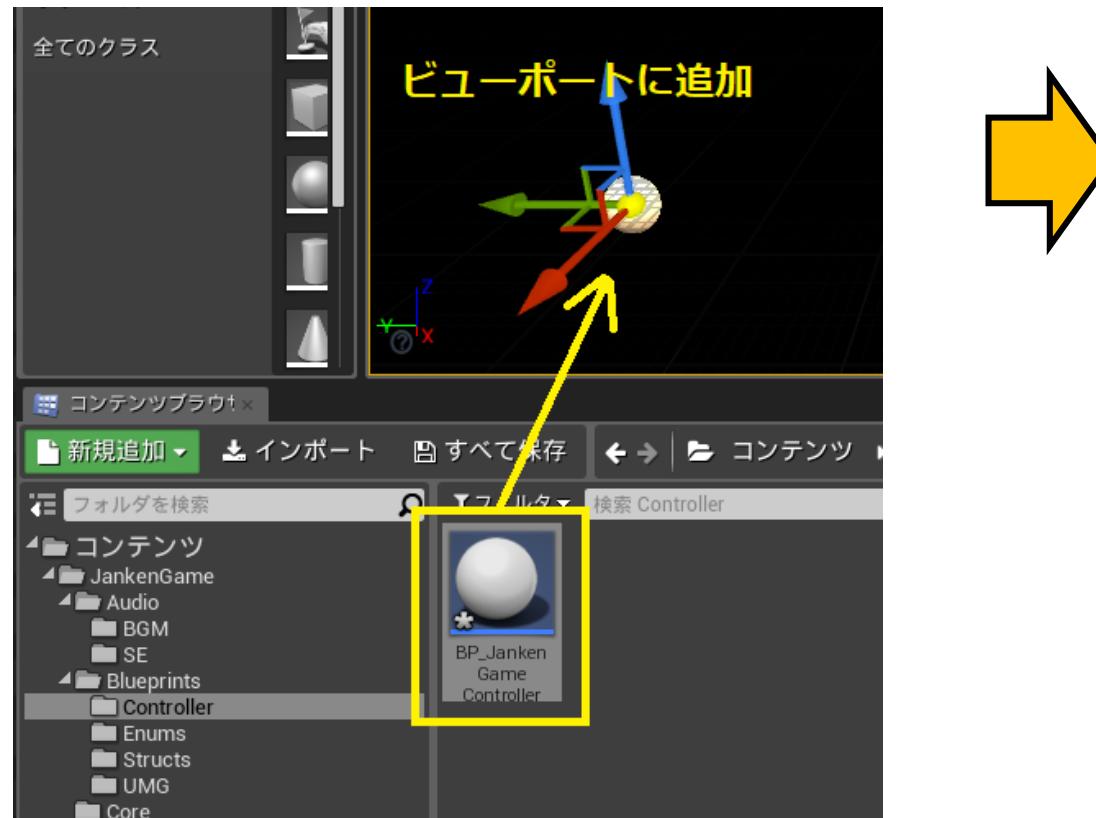


右クリック > 0で検索
> キーボードイベント 0を選択

右クリック > printstringで検索
> printstringを選択

In Stringに[グー]を設定
PressedとPrintStringをつなぐ

ビューポートにブループリントを追加
0キーをPressでグーが出力されることを確認



ビューポートにBP_JankenGameControllerを追加

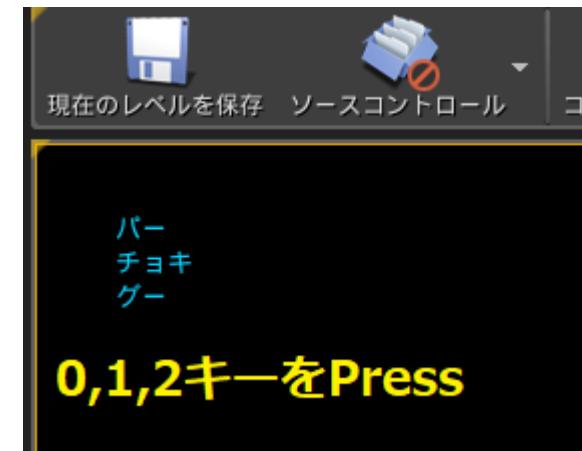


プレイして確認
0キーをPressでグーが出力される

キーボード 1,2をPressed時にチョキ、
パーを出力



1キー Pressed : チョキを出力
2キー Pressed : パーを出力



プレイして確認
0,1,2キーをPressして出力確認

3.じゃんけんゲームの作成

3.1 新規レベル（Game）を作成・保存

3.2 ブループリント(BP_JankenGameController)を作成

3.3 簡単なじゃんけんゲームを実装

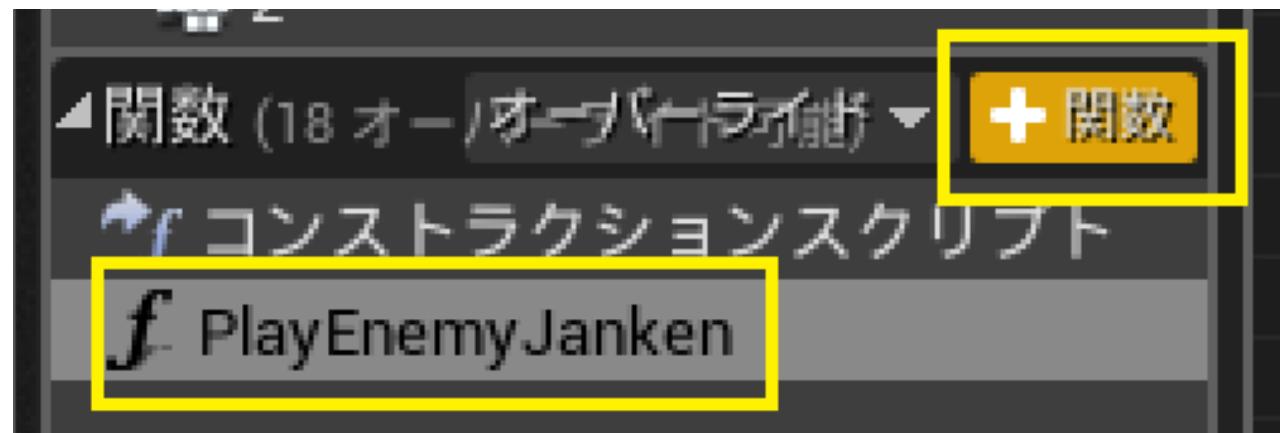
3.3.1 プレイヤーはキーボード入力でグー、チョキ、パーを出力

3.3.2 関数:PlayEnemyJanken (敵はランダムでグー、チョキ、パー)

3.3.3 プレイヤーと敵のじゃんけん結果から勝敗を判定する
関数:JudgeJankenを作成

3.3.4 作成した関数を使ってじゃんけんゲームを実装

関数:PlayEnemyJankenを作成

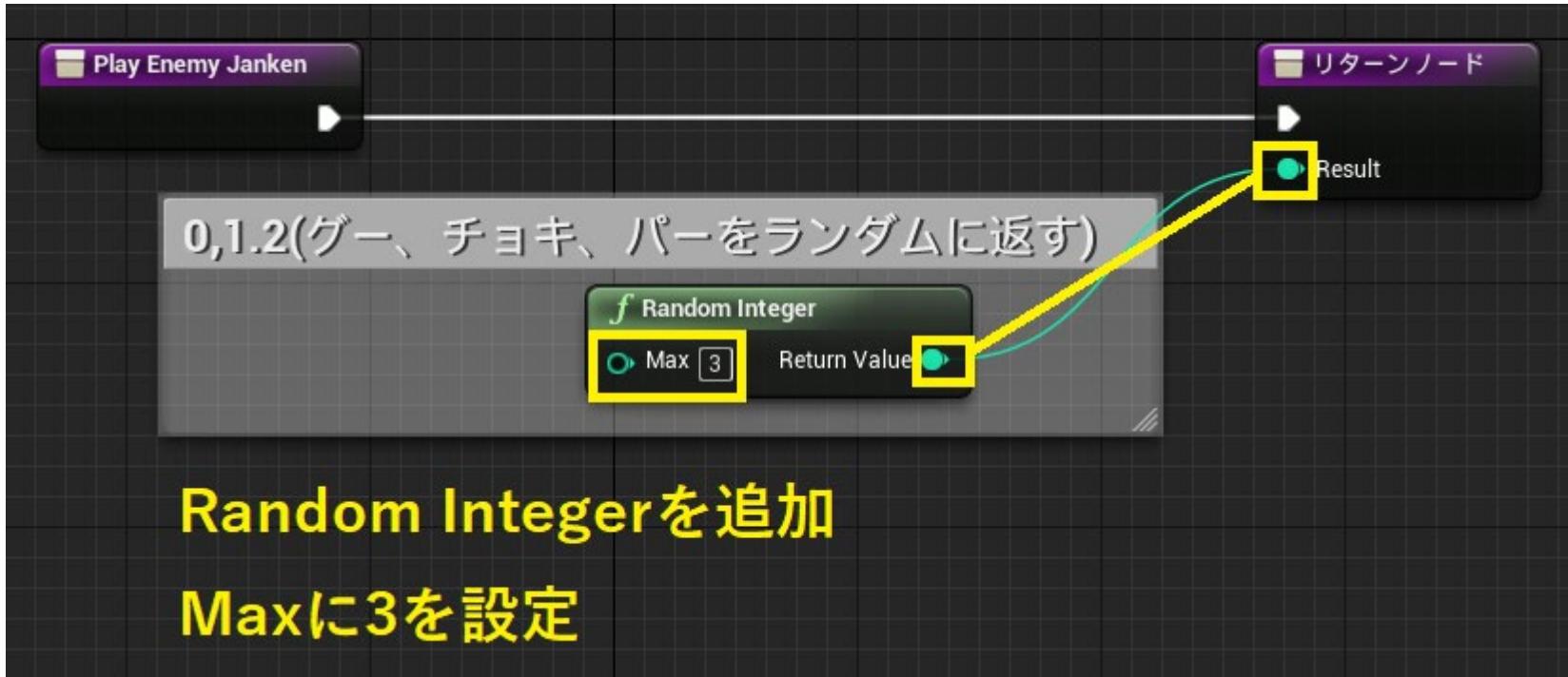


関数 : PlayEnemyJankenを追加する

関数名	アクセス指定子	Input/Output	パラメータ名	型
PlayEnemyJanken	プライベート	Output	Result	Integer

説明 : 敵のじゃんけん結果を返す
(※Publicの時は説明を書いておこう)

ランダムで0,1,2を返す
Random Integerノードを使用する



Random Integerを追加
Maxに3を設定

3.じゃんけんゲームの作成

3.1 新規レベル（Game）を作成・保存

3.2 ブループリント(BP_JankenGameController)を作成

3.3 簡単なじゃんけんゲームを実装

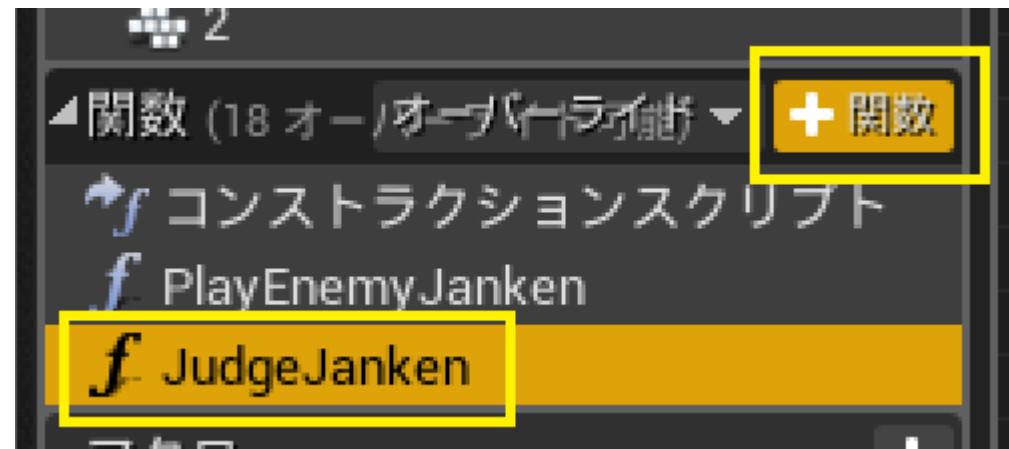
3.3.1 プレイヤーはキーボード入力でグー、チョキ、パーを出力

3.3.2 関数:PlayEnemyJanken (敵はランダムでグー、チョキ、パー)

3.3.3 プレイヤーと敵のじゃんけん結果から勝敗を判定する
関数:JudgeJankenを作成

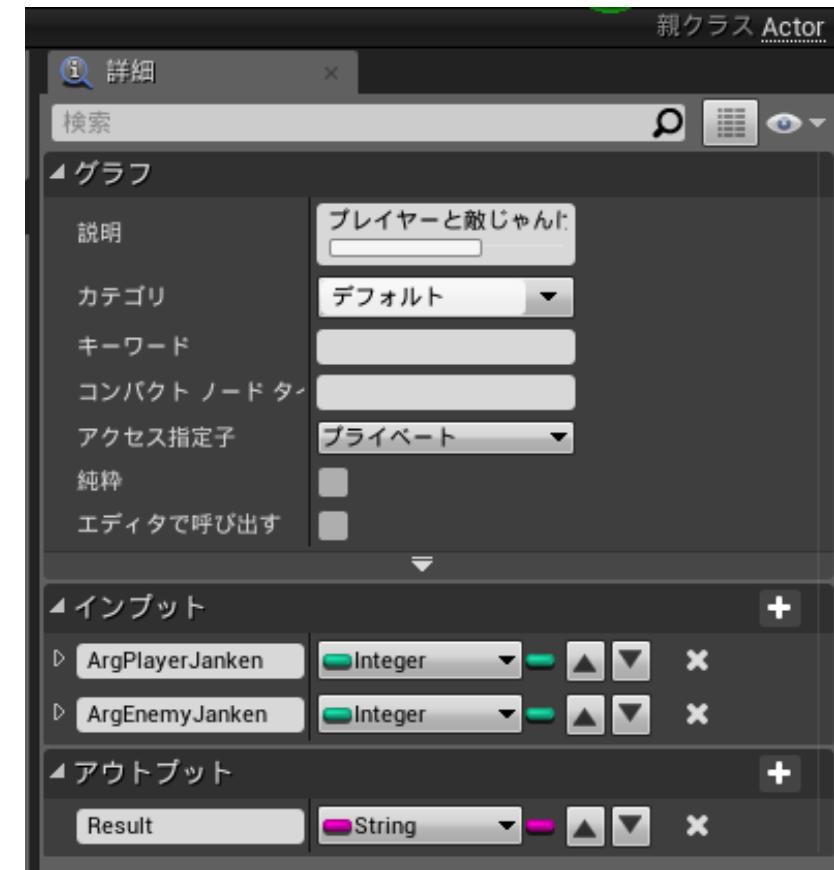
3.3.4 作成した関数を使ってじゃんけんゲームを実装

プレイヤーと敵のじゃんけん結果から勝敗を判定する関数:JudgeJankenを作成



関数：JudgeJankenを追加する

関数名	アクセス指定子	Input/Output	パラメータ名	型
JudgeJanken	プライベート	Input	ArgPlayerJanken	Integer
		Input	ArgEnemyJanken	Integer
		Output	Result	String

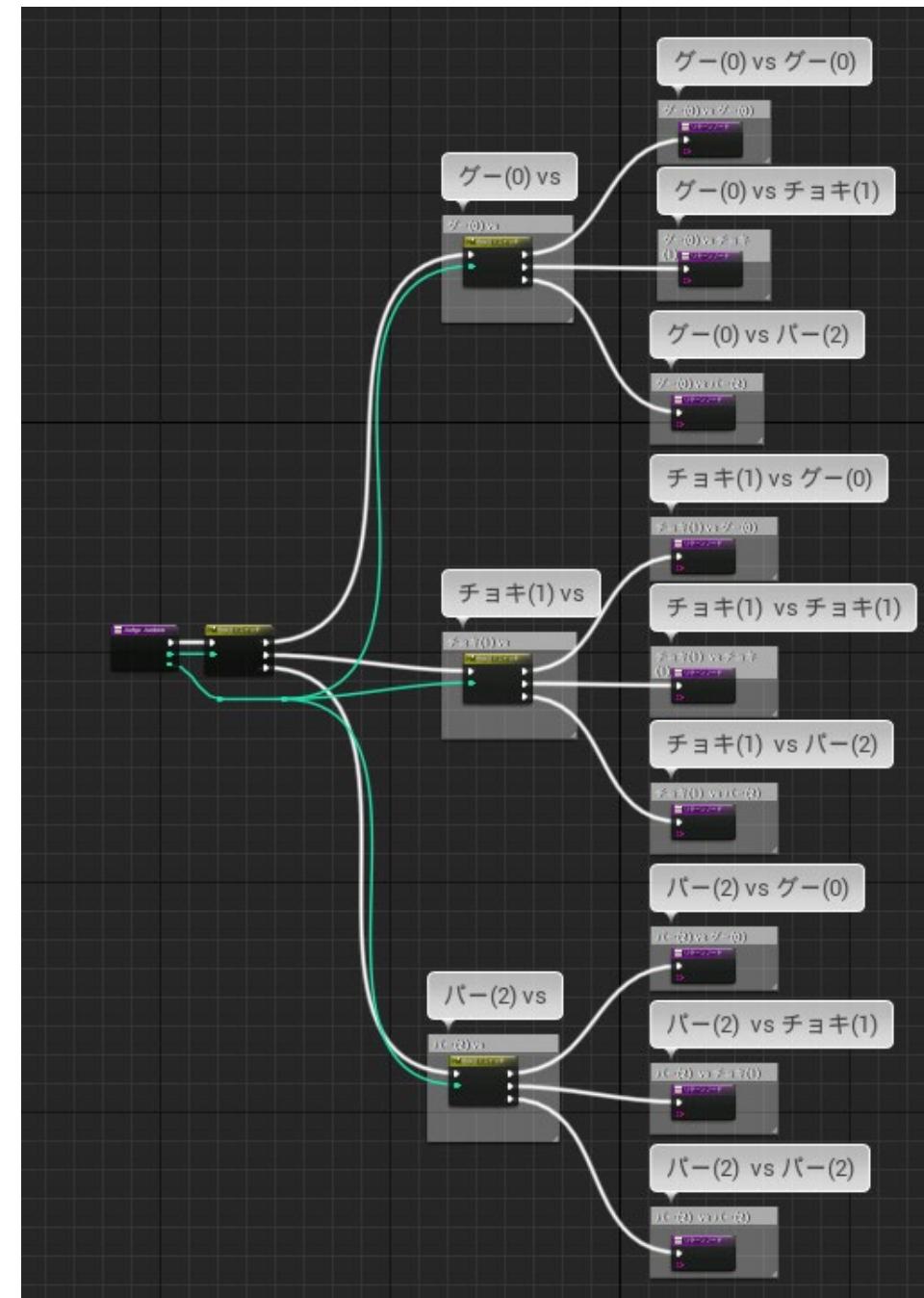


説明：プレイヤーと敵じゃんけん結果から勝敗を判定する

スイッチノードで
じゃんけんの勝敗を
リターンノードに設定

リターンノードに設定する文字列

Player \ Enemy	グー(0)	チョキ(1)	パー(2)
グー(0)	Draw	Win	Lose
チョキ(1)	Lose	Draw	Win
パー(2)	Win	Lose	Draw



Arg Player Jankenを分岐するSwitchノードを追加



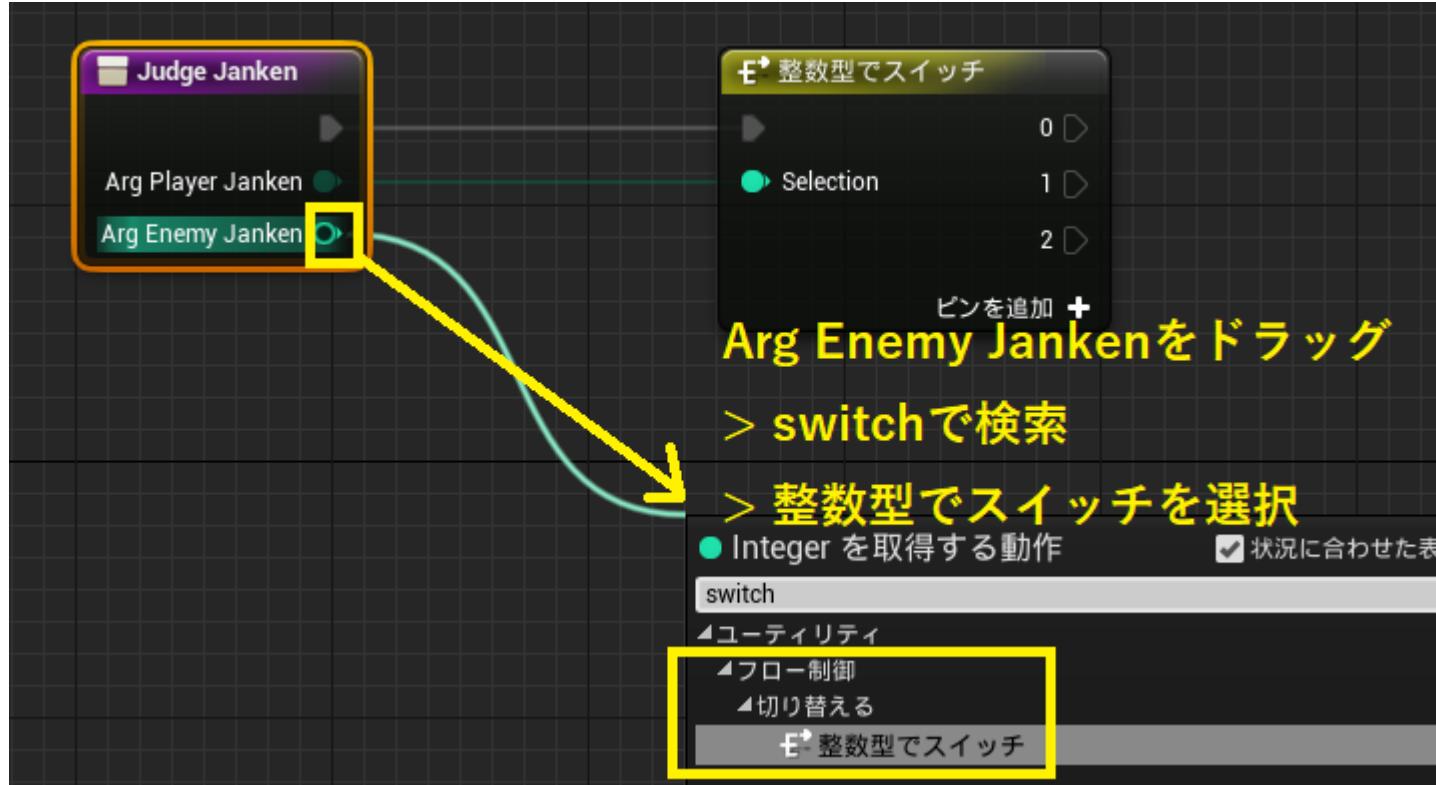
Arg Player Jankenをドラッグ > switchで検索 > 整数型でスイッチを選択

スイッチノードの設定を行う



スイッチノードのHas Default Pinのチェックを外す
[Pinを追加]をクリックして、0~2で分岐するように設定

Arg Enemy Jankenを分岐するSwitchノードを追加



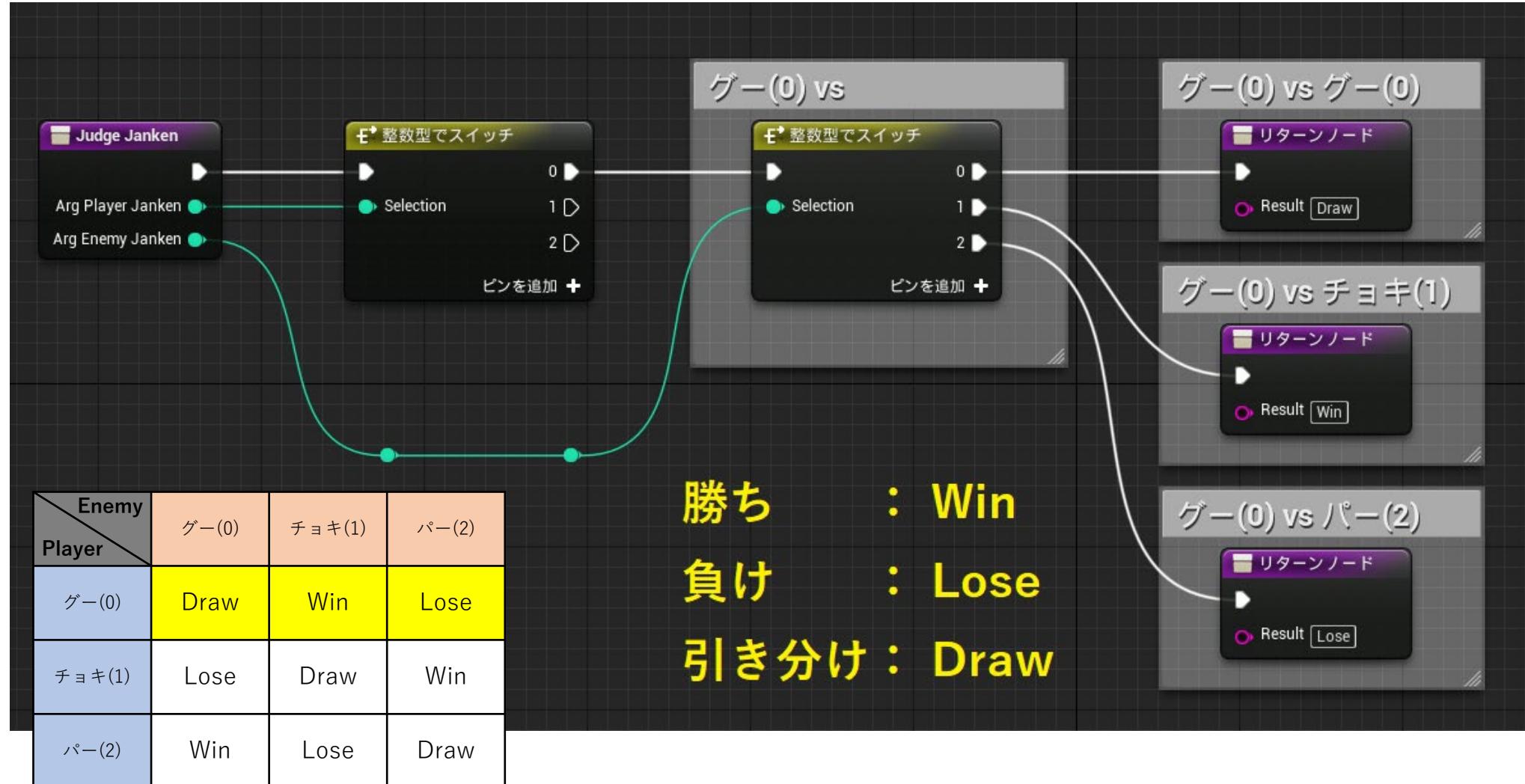
Arg Enemy Jankenをドラッグ > switchで検索 > 整数型でスイッチを選択

スイッチノードの設定を行う

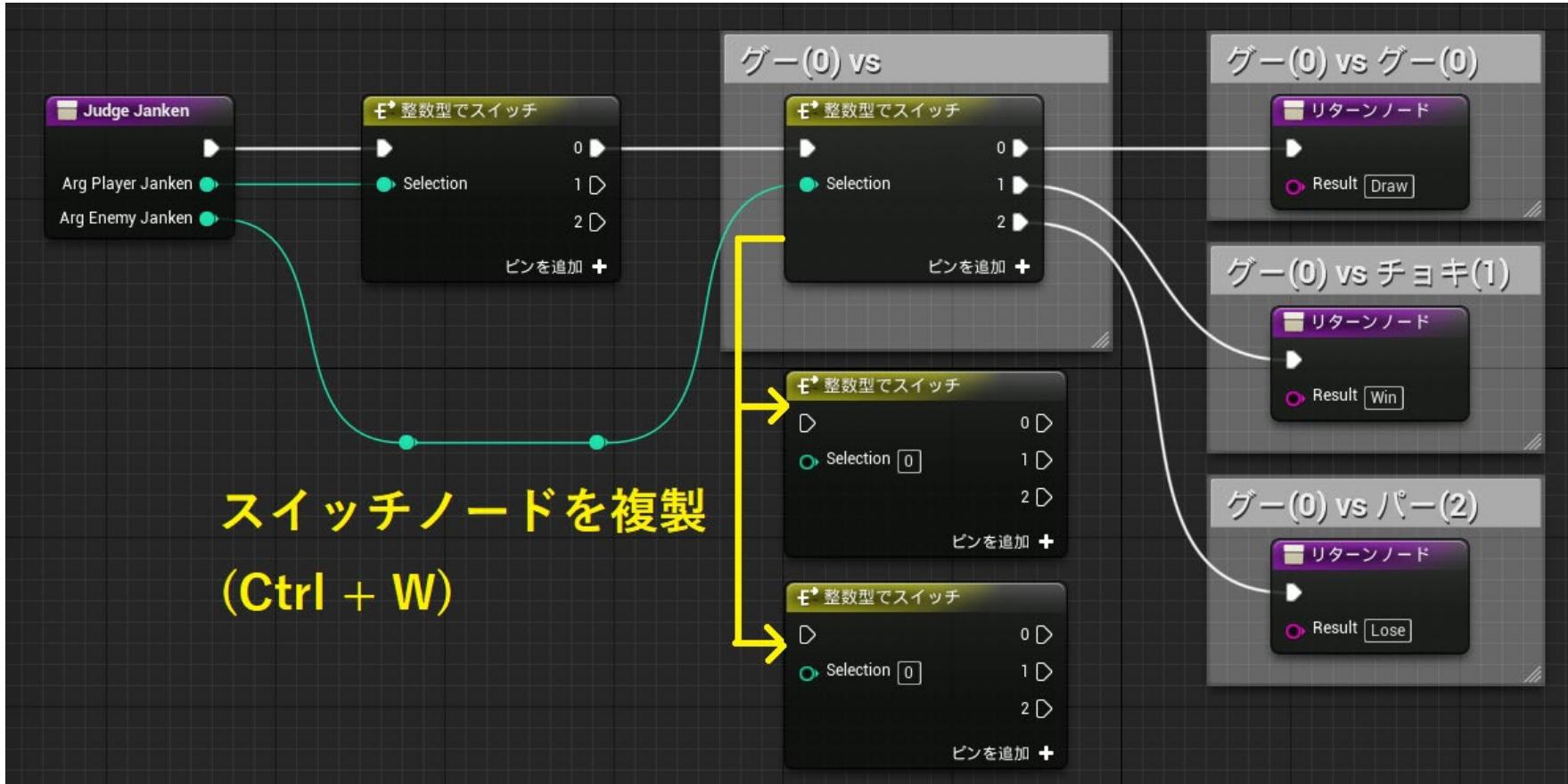


スイッチノードのHas Default Pinのチェックを外す
[ピンを追加]をクリックして、0~2で分岐するように設定

プレイヤーがグー(0)の時の比較を行い リターンノードに結果の文字列を設定

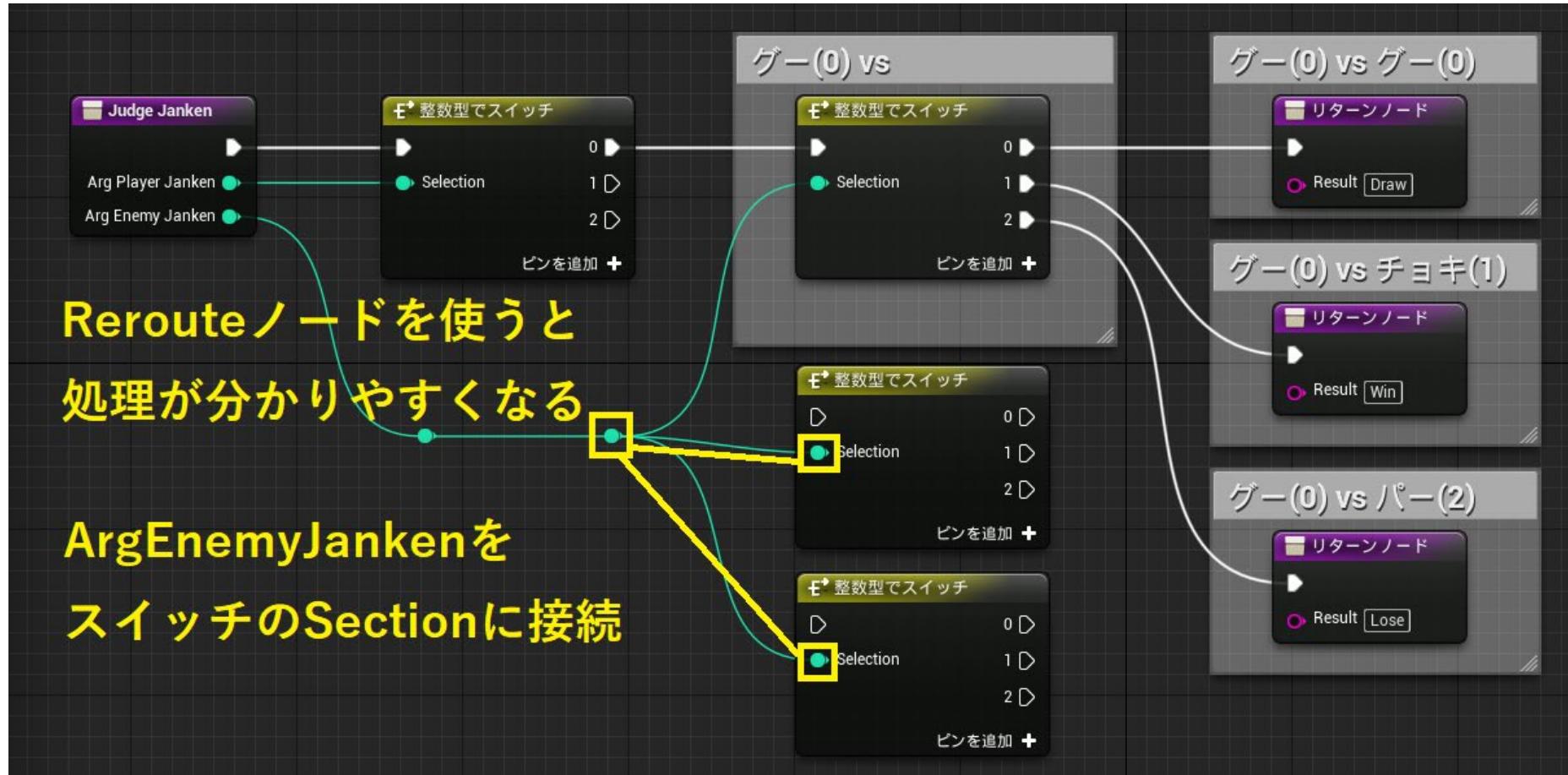


Enemy側のスイッチを複製 (プレイヤーがチョキ、パーの場合の分岐)



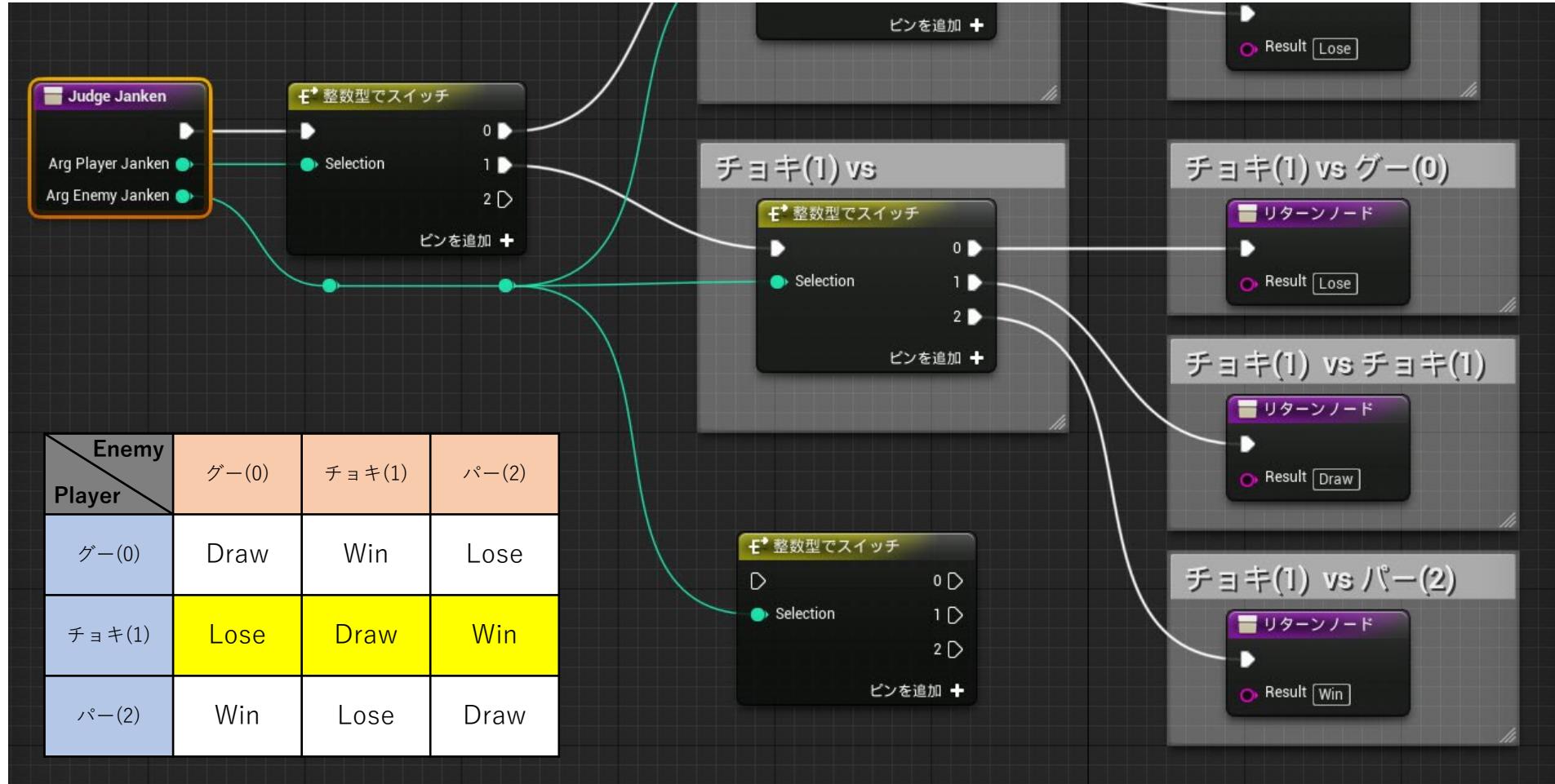
Enemy側のスイッチを複製(Ctrl + Wで複製)

スイッチノードのSectionにEnemyJankenを接続

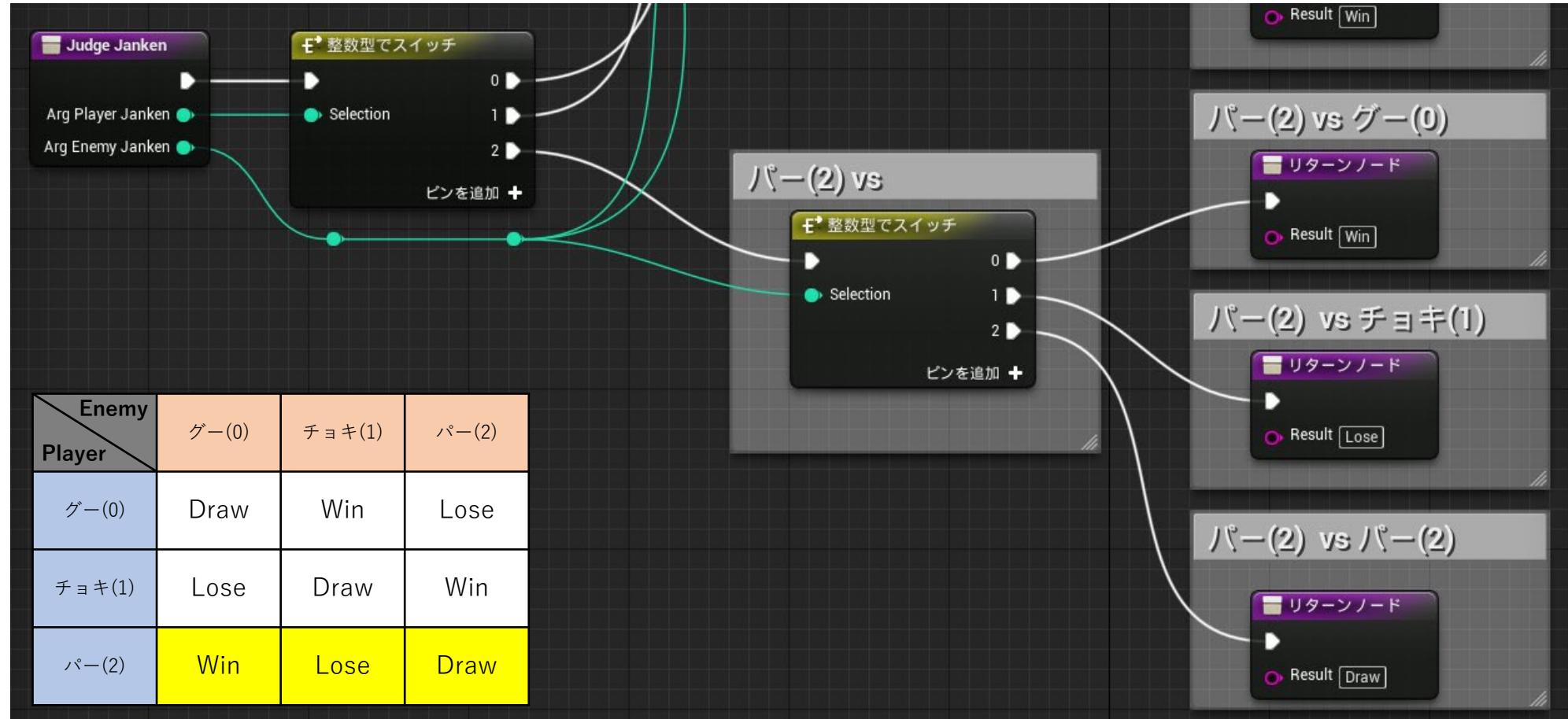


SwitchノードのSectionにEnemyJankenを接続

プレイヤーがチョキ(1)の時の勝敗をリターンノードに設定する



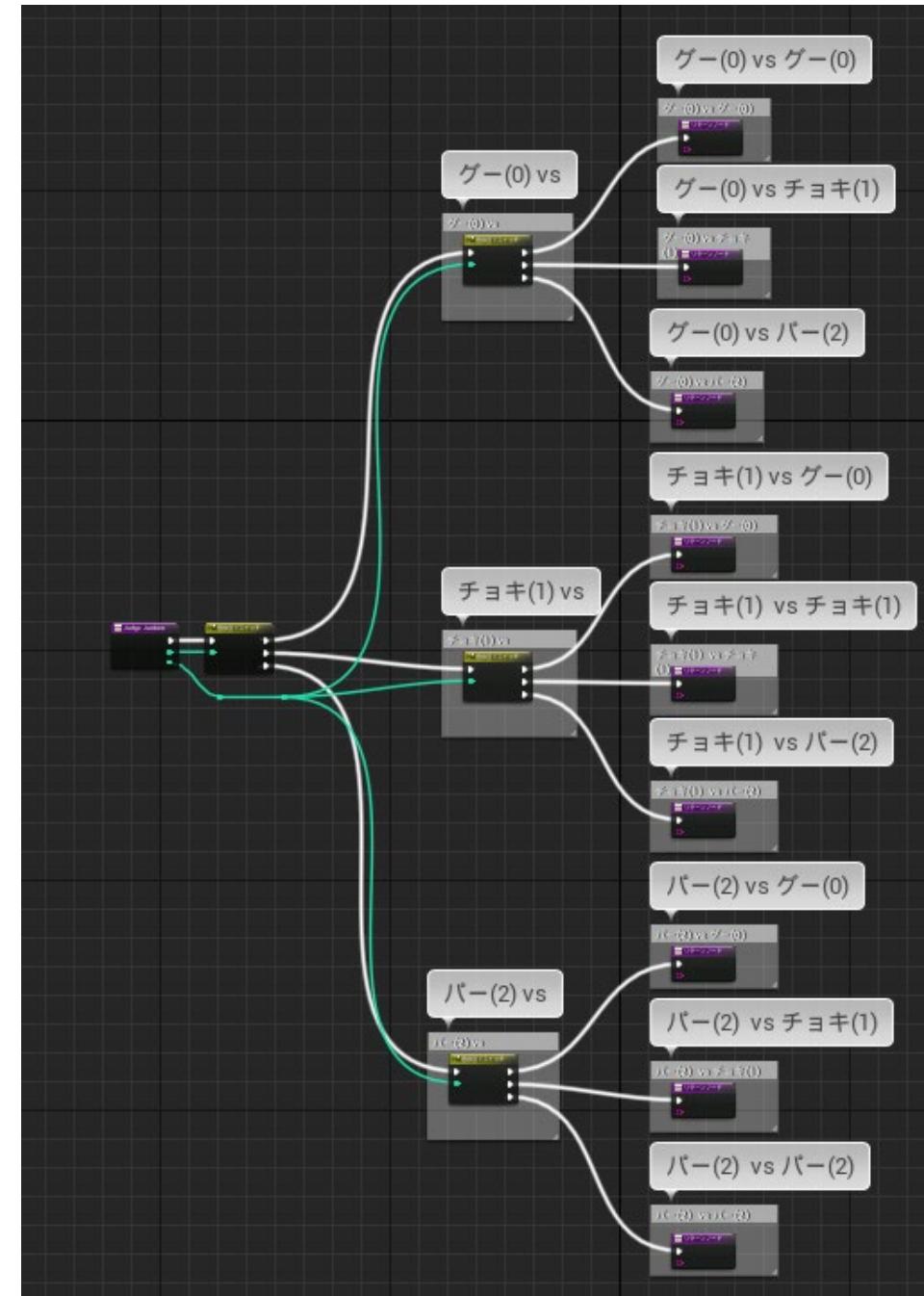
プレイヤーがパー(2)の時の勝敗を リターンノードに設定する



スイッチノードで
じゃんけんの勝敗を
リターンノードに設定

リターンノードに設定する文字列

Player \ Enemy	グー(0)	チョキ(1)	パー(2)
グー(0)	Draw	Win	Lose
チョキ(1)	Lose	Draw	Win
パー(2)	Win	Lose	Draw



3.じゃんけんゲームの作成

3.1 新規レベル（Game）を作成・保存

3.2 ブループリント(BP_JankenGameController)を作成

3.3 簡単なじゃんけんゲームを実装

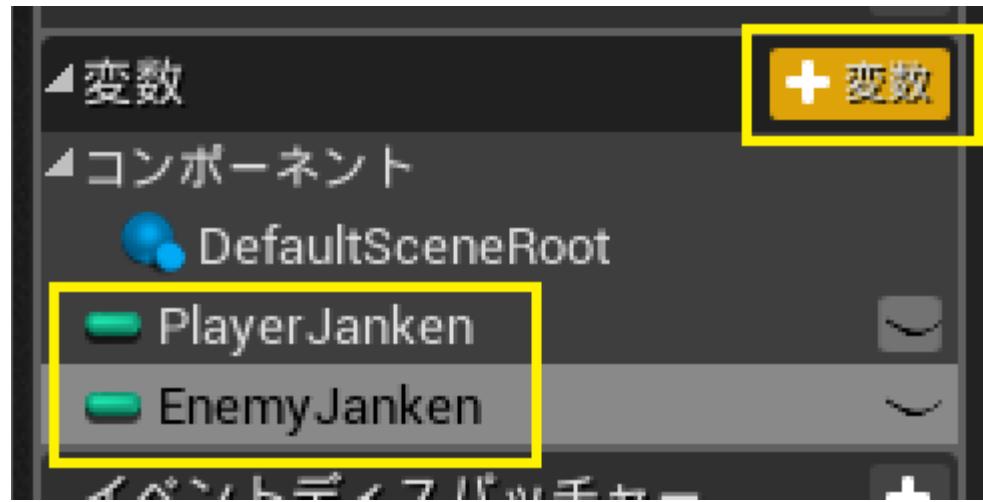
3.3.1 プレイヤーはキーボード入力でグー、チョキ、パーを出力

3.3.2 関数:PlayEnemyJanken (敵はランダムでグー、チョキ、パー)

3.3.3 プレイヤーと敵のじゃんけん結果から勝敗を判定する
関数:JudgeJankenを作成

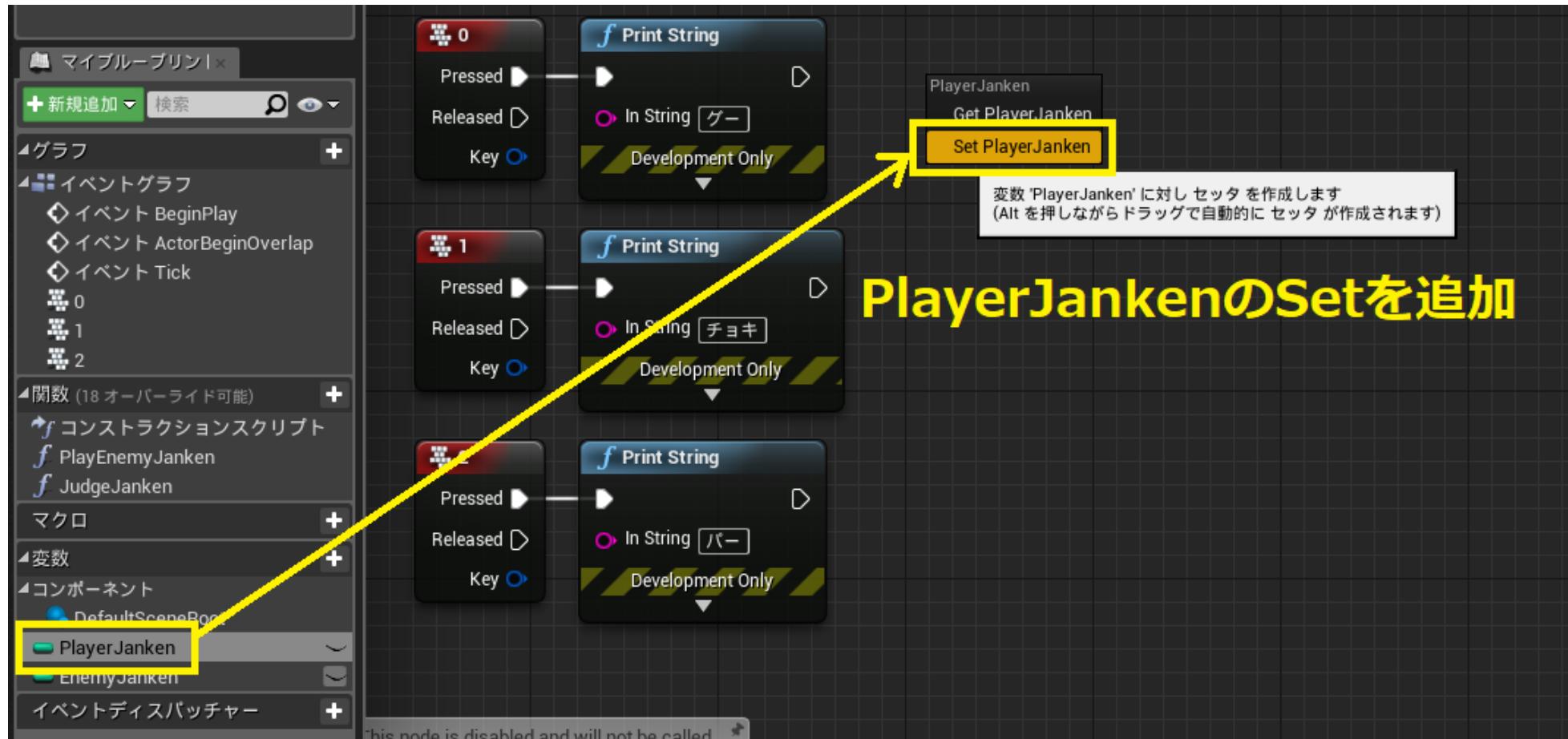
3.3.4 作成した関数を使ってじゃんけんゲームを実装

変数を設定



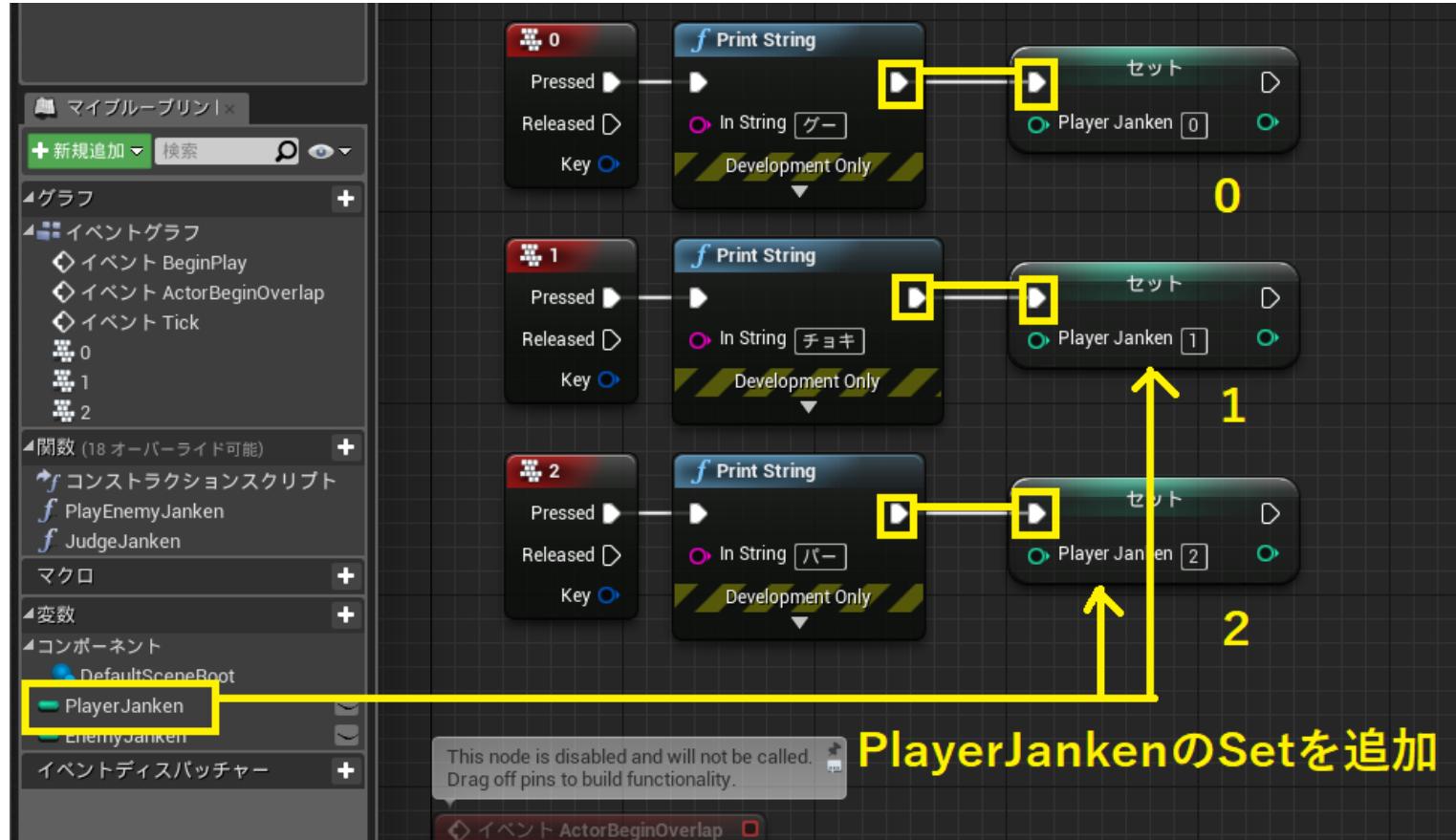
変数名	変数の型	デフォルト値
PlayerJanken	Integer	-(設定なし)
EnemyJanken	Integer	-(設定なし)

PlayerJankenのSetを追加



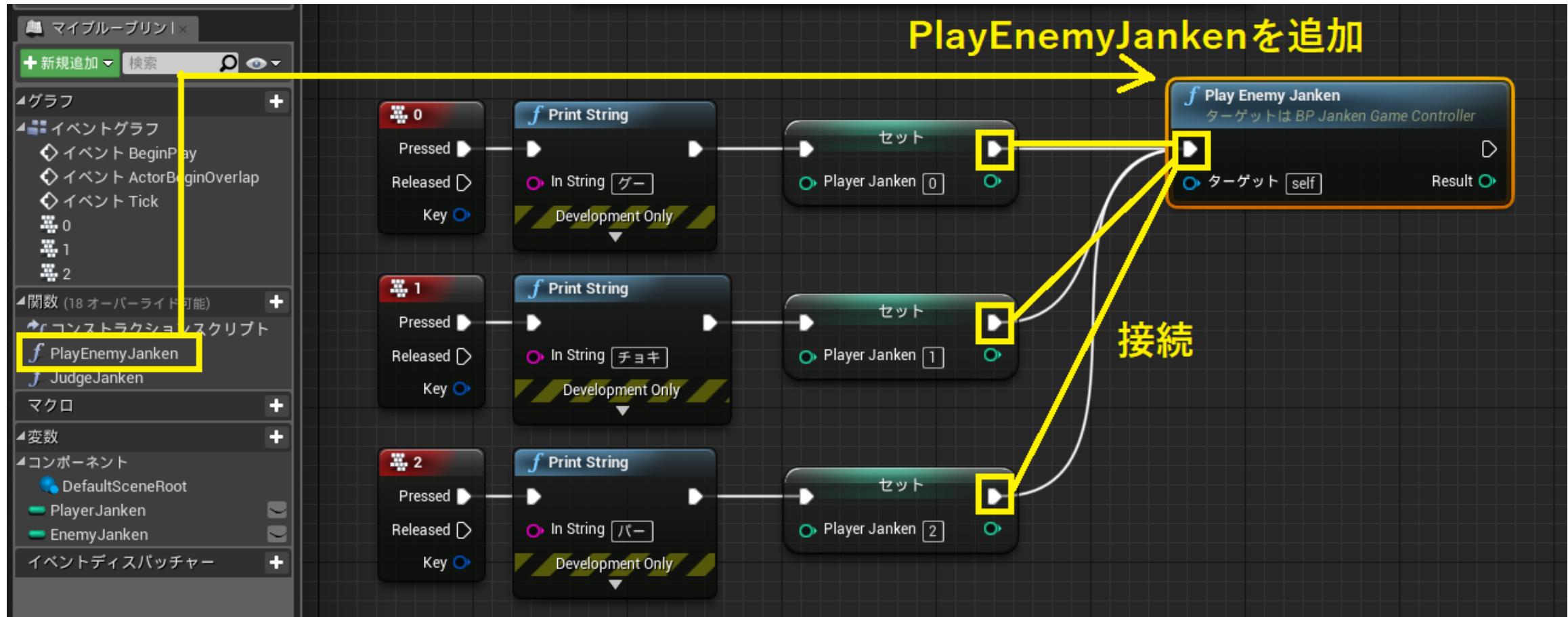
PlayerJankenをGraphにドラッグ > SetPlayerJankenを選択

キー ボー ド1,2(Pressed)にもPlayerJankenのSetを追加
値を設定(グー:0 チョキ:1 パー:2)



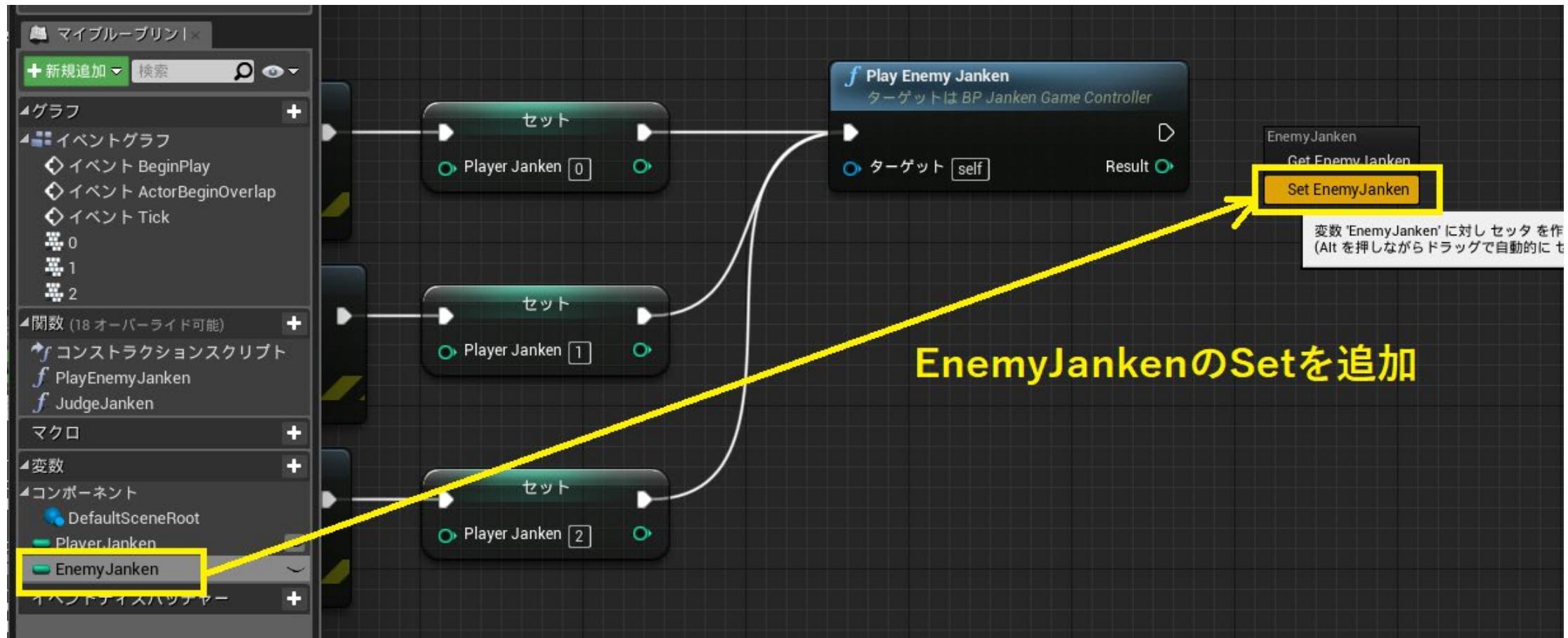
PlayerJankenのSetを追加 > 値を設定 > Print Stringと各セットノードと接続

PlayEnemyJankenを追加
キー ボード入力のセットを PlayEnemyJanken に接続



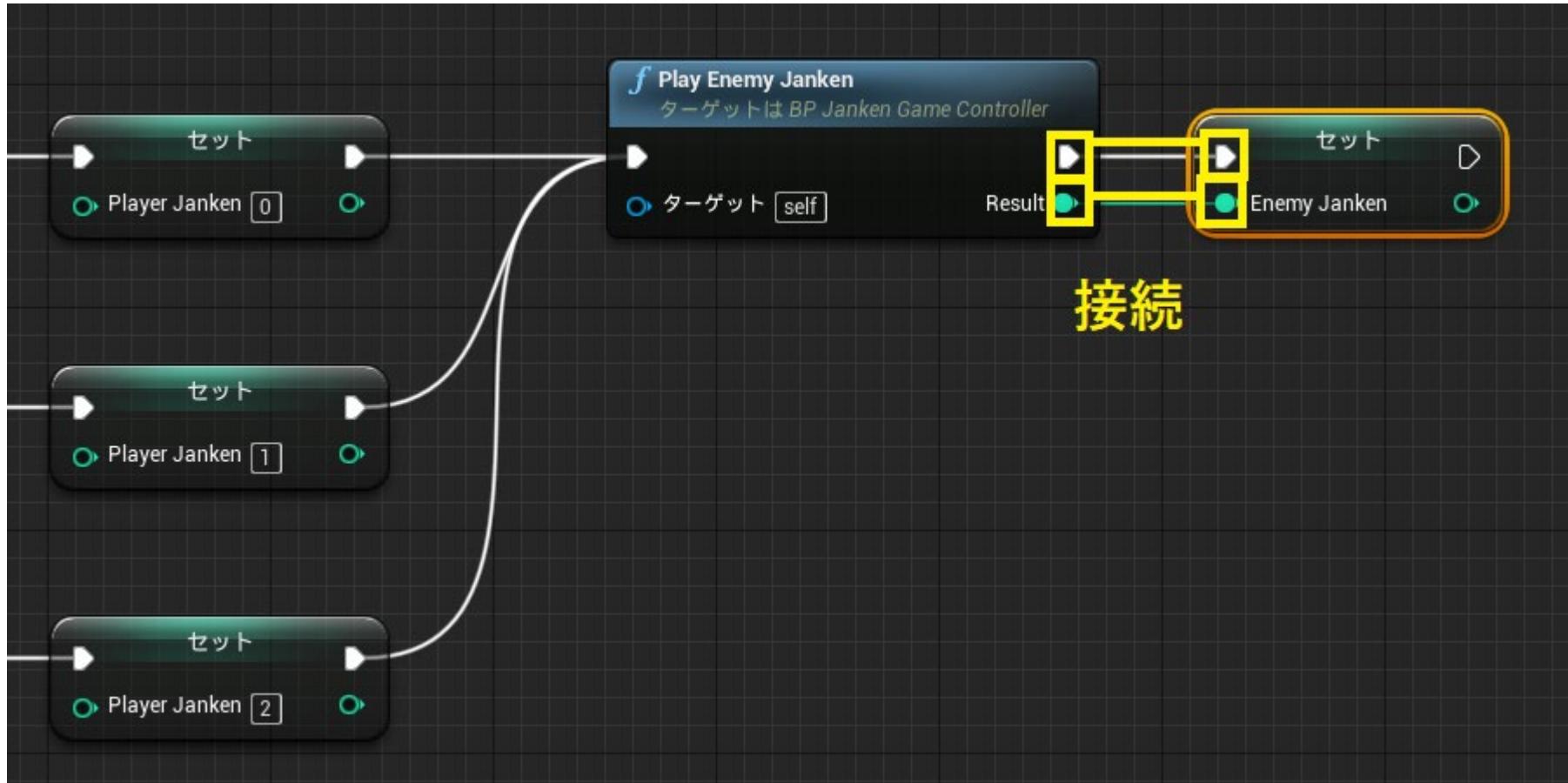
PlayEnemyJankenを追加 > PlayerJankenのセットとPlayEnemyJankenを接続

EnemyJankenのSetを追加



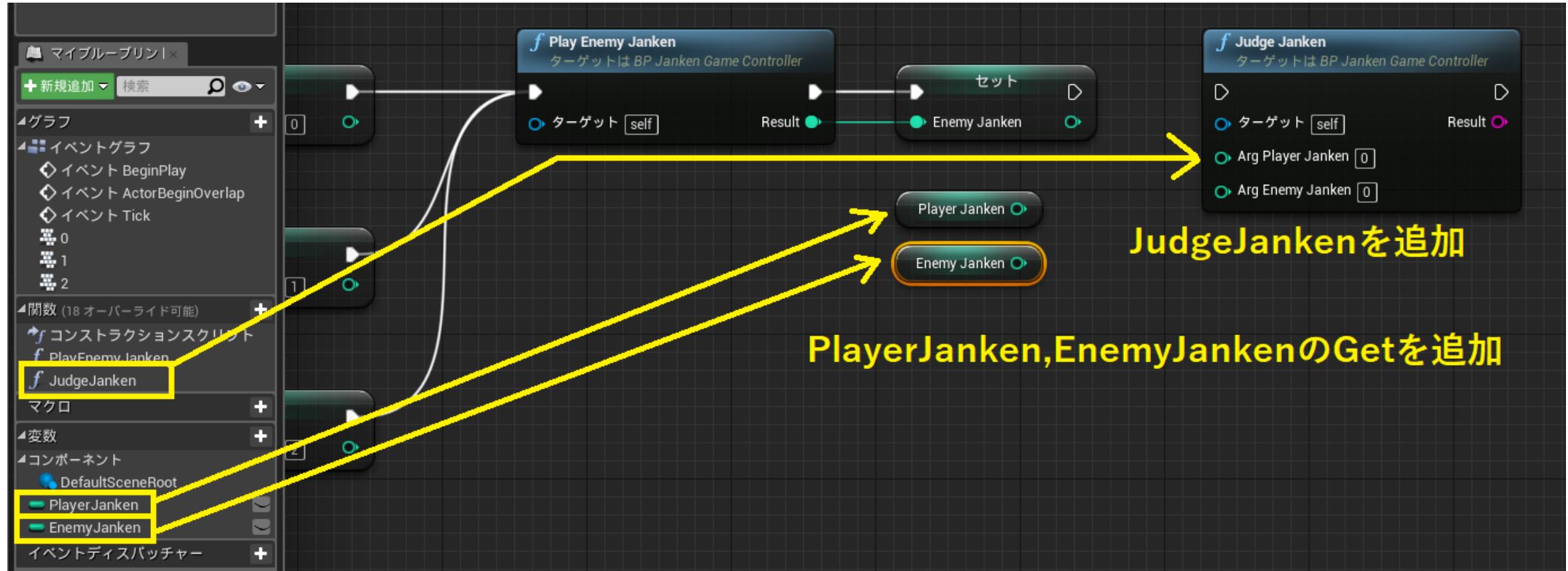
EnemyJankenをGraphにドラッグ > SetEnemyJankenを選択

PlayEnemyJankenとEnemyJankenを接続



PlayEnemyJankenとEnemyJankenのSetを接続

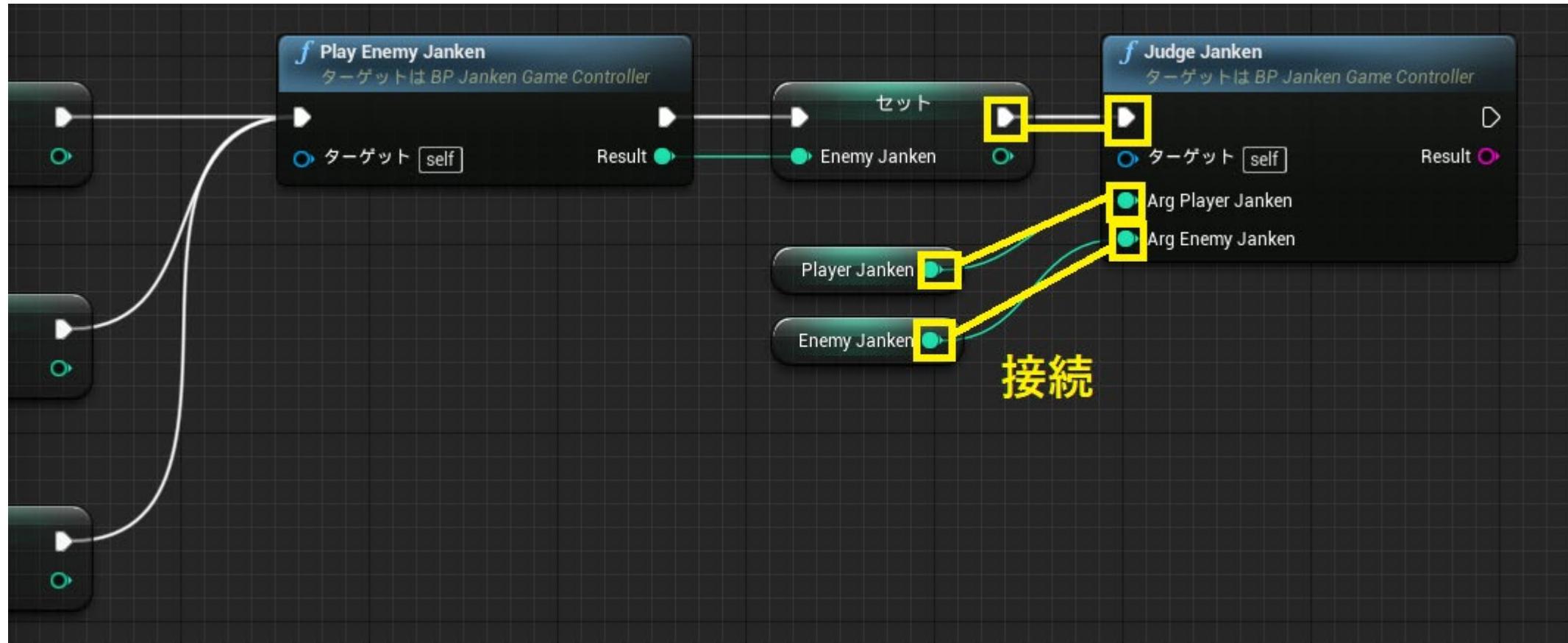
JudgeJankenを追加 PlayerJanken,EnenmyJankenのGetを追加



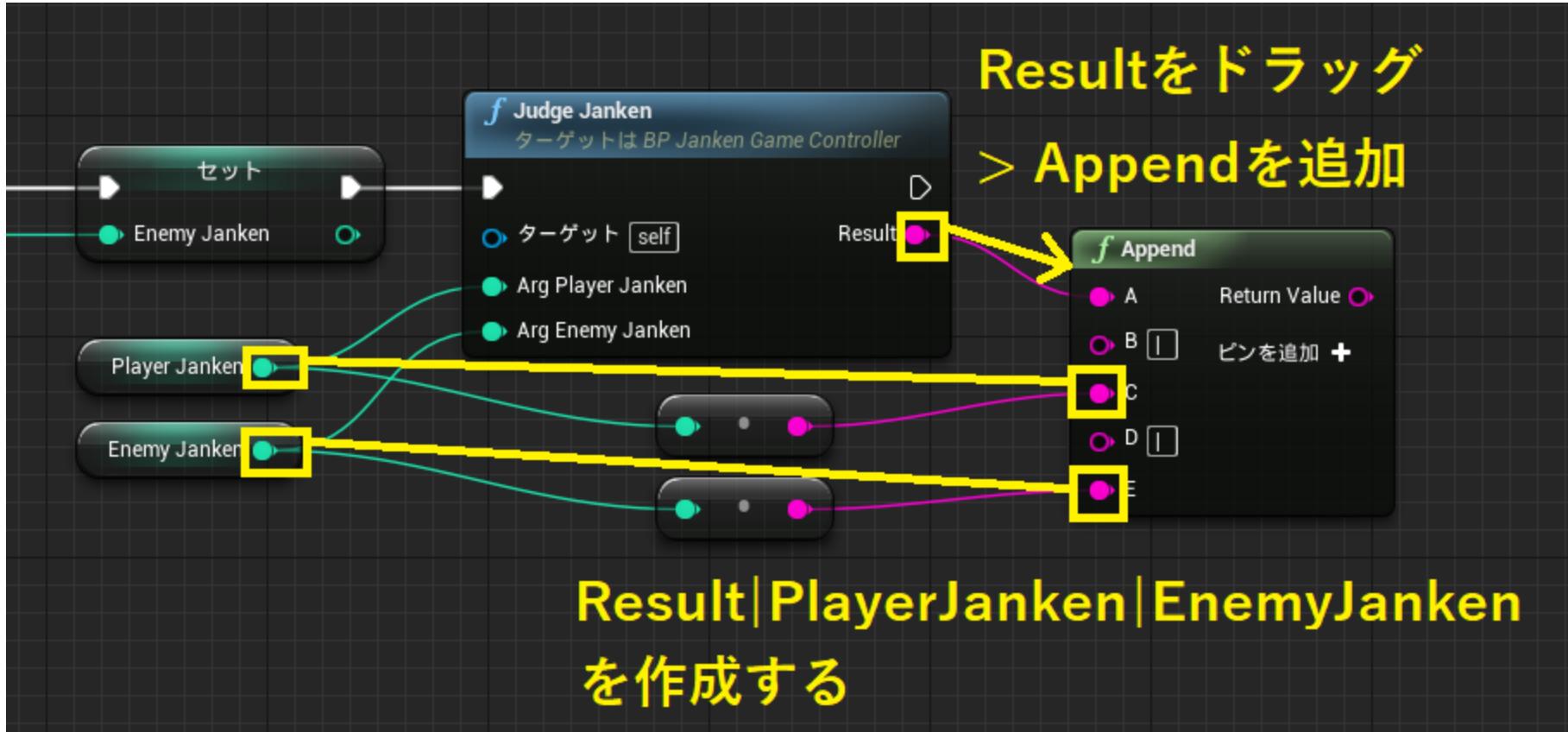
JudgeJankenを追加

PlayerJanken,EnenmyJankenのGetを追加

JudgeJankenの実行、インプットに接続

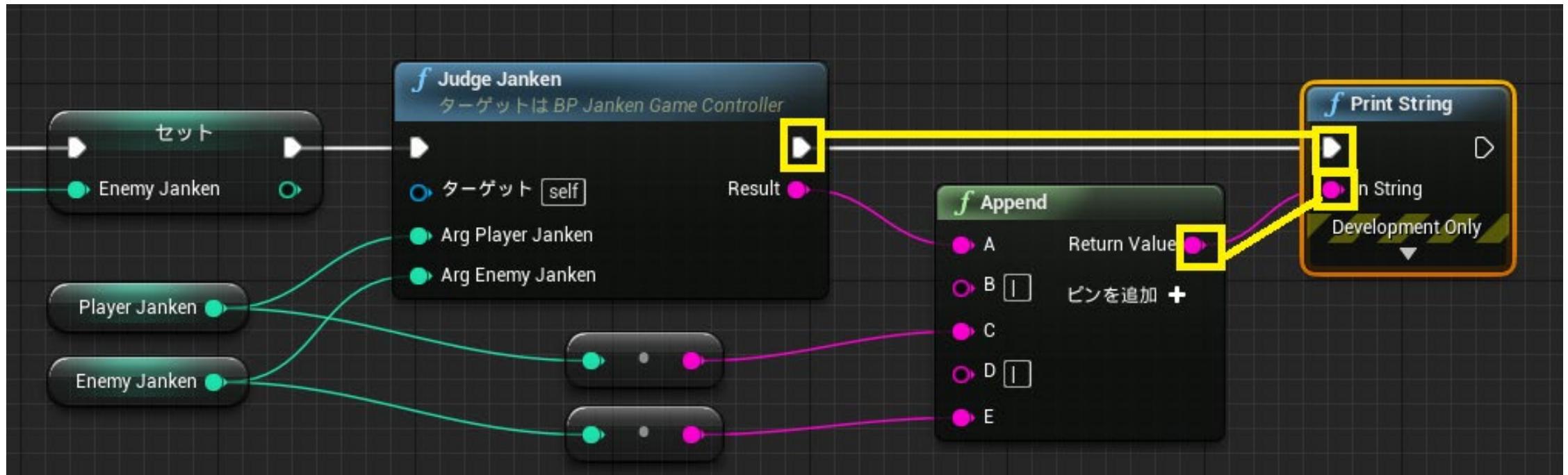


Appendノードを追加
文字列を作成：Result|PlayerJanken|EnemyJanken



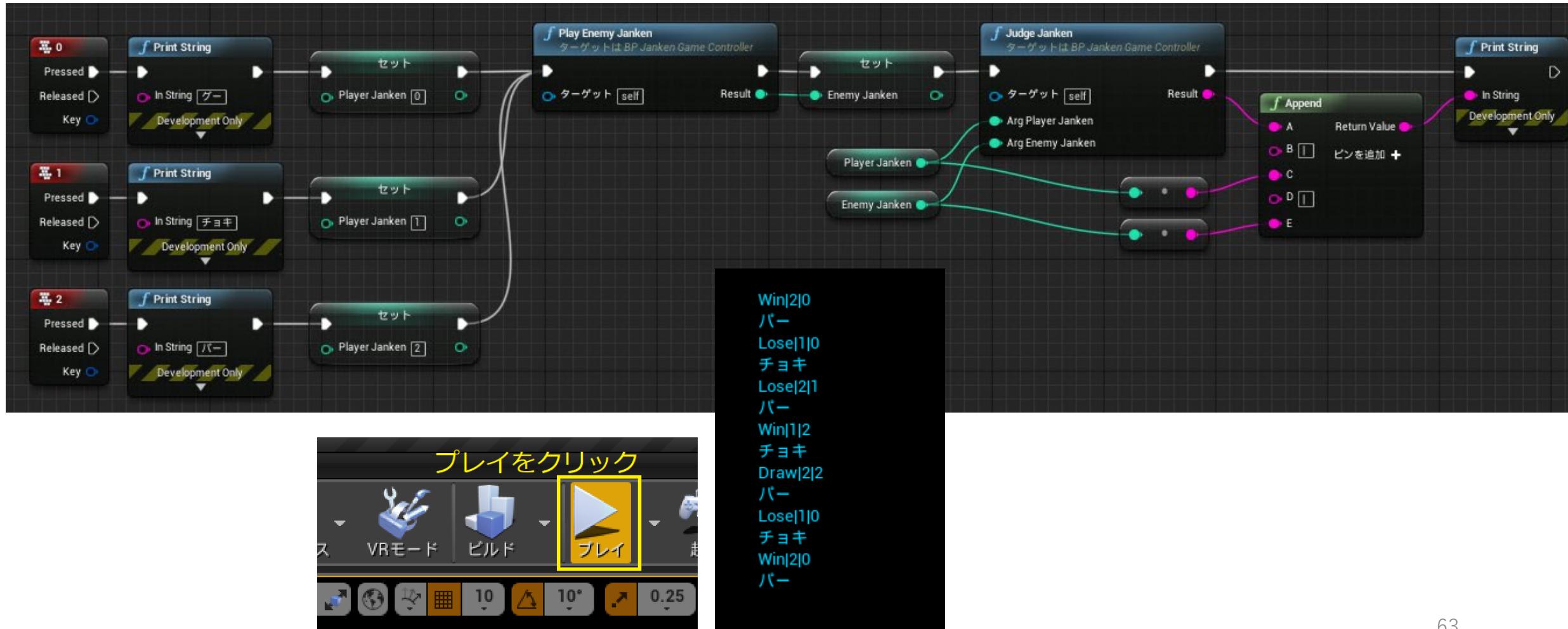
Appendノードを追加
文字列を作成：Result|PlayerJanken|EnemyJanken

ApendノードのReturnValueをPrintStringで出力



ApendノードのReturnValueをPrintStringで出力

プレイして確認



4. 列挙型を使ってエラーが少ない処理にする

4. 列挙型を使ってエラーが少ない処理にする

4.1 列挙型:EJankenを作成

4.2 列挙型:EJankenResultを作成

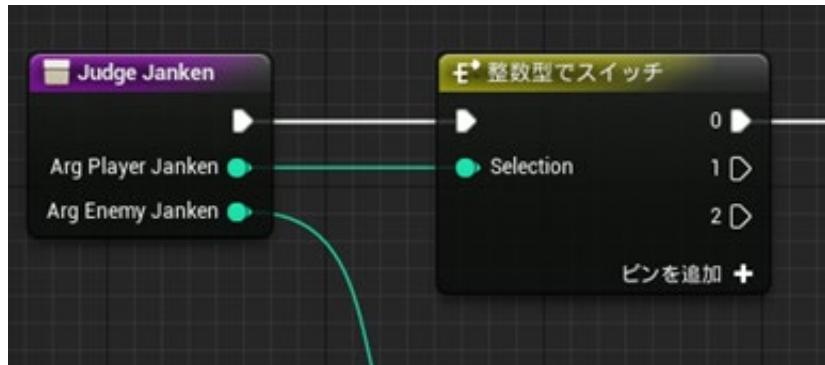
4.3 関数:PlayEnemyJankenのアウトプットの型をEJankenに変更

4.4 関数: JudgeJankenのインプットの型をEjanken,
アウトプットの型をEJankenResultに変更

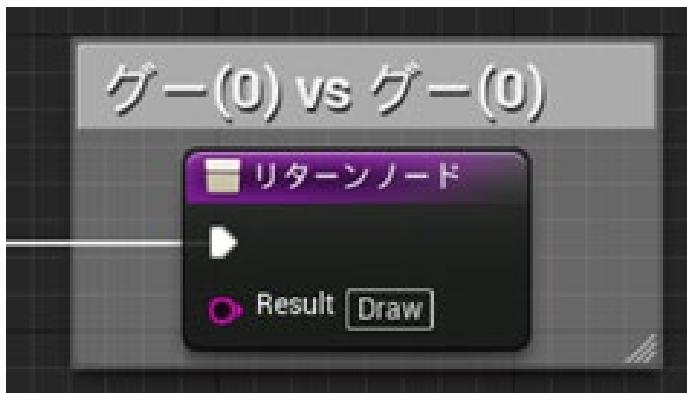
4.5 関数のインプット・アウトプットの変更によるエラー修正

列挙型を使用することで処理を見やすく、エラーを少なくする

ここまで

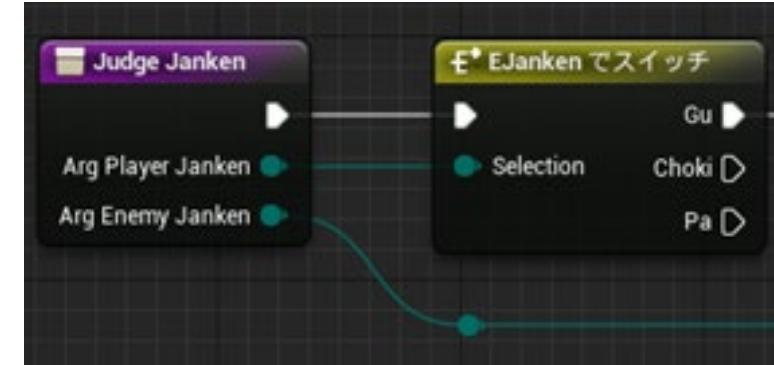


分岐の数値が何を表すか分からぬ



文字列の手入力でリターンノードを返している

列挙型を使用した



何の値で分岐をしているか文字列を表示できる



リターンノードに設定する値をリストから選択

4. 列挙型を使ってエラーが少ない処理にする

4.1 列挙型:EJankenを作成

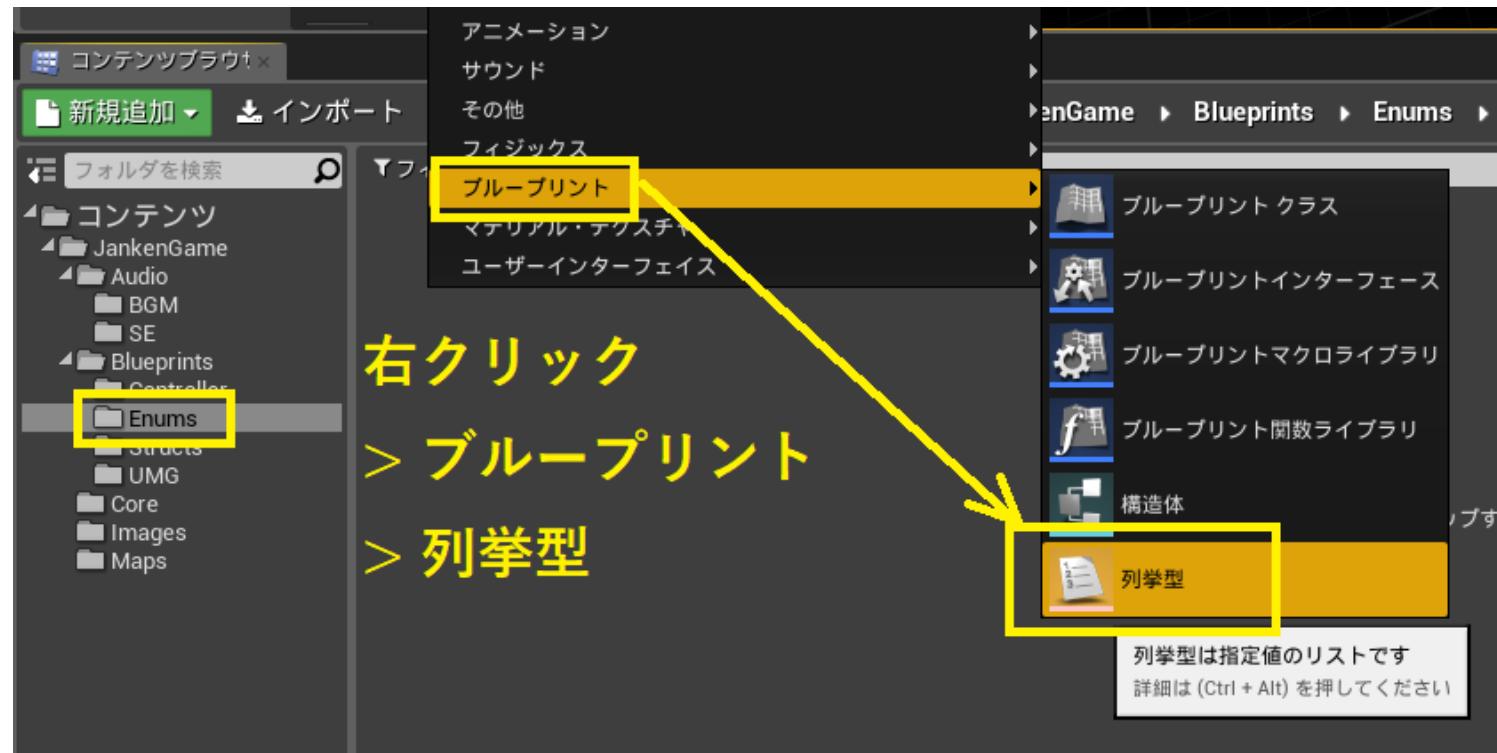
4.2 列挙型:EJankenResultを作成

4.3 関数:PlayEnemyJankenのアウトプットの型をEJankenに変更

4.4 関数: JudgeJankenのインプットの型をEjanken,
アウトプットの型をEJankenResultに変更

4.5 関数のインプット・アウトプットの変更によるエラー修正

列挙型:EJankenを作成

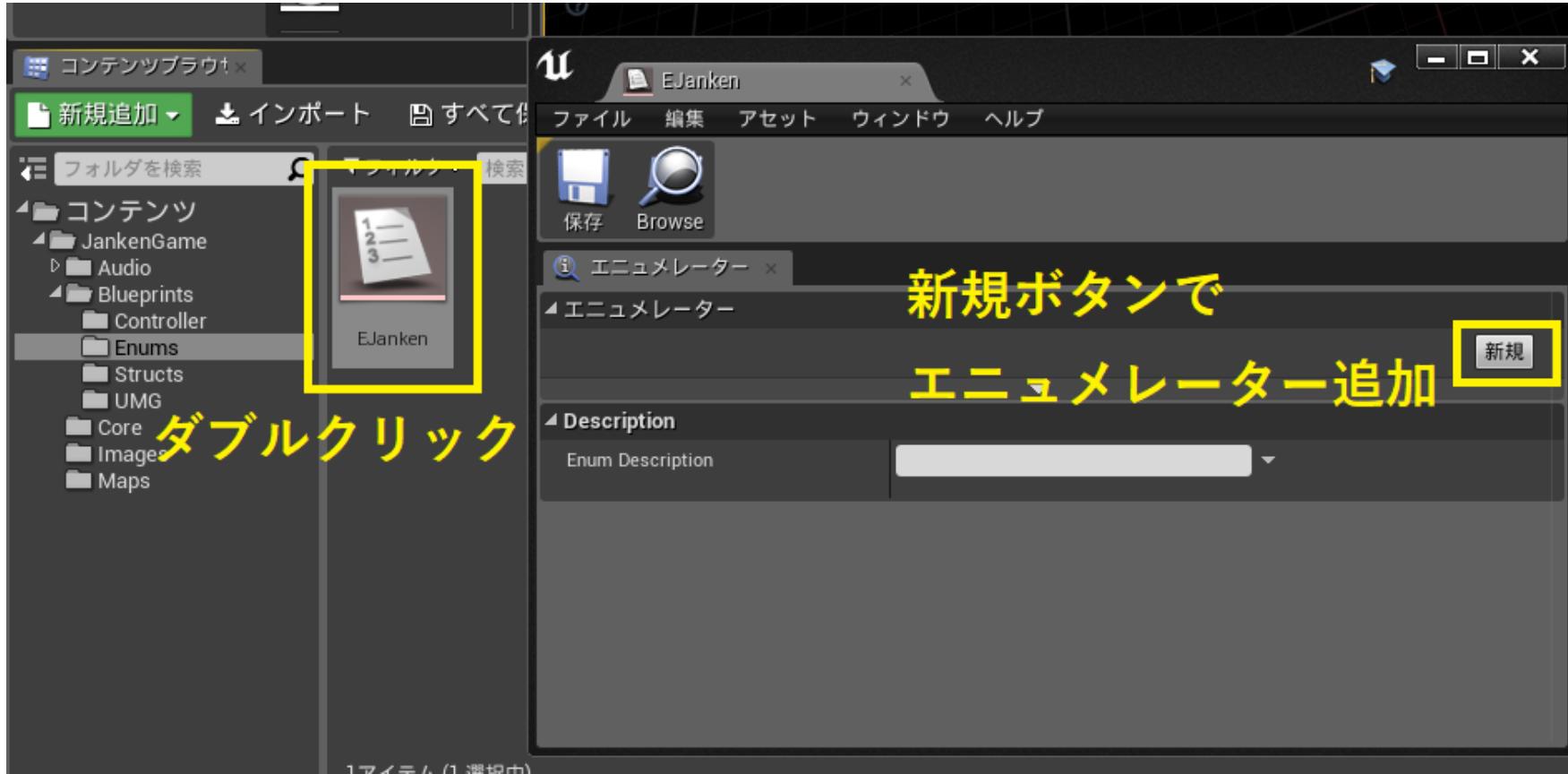


右クリック > ブループリント > 列挙型



名前をEJankenに設定

Ejankenを開く 新規ボタンでエニュメレーターを追加



Ejankenを開く
新規ボタンでエニュメレーターを追加

エニュメレーターを設定 Description(説明)も設定



エニュメレーター

Display Name	Description
Gu	グー
Choki	チョキ
Pa	パー

Description

Enum Description

じゃんけんの種別

4. 列挙型を使ってエラーが少ない処理にする

4.1 列挙型:EJankenを作成

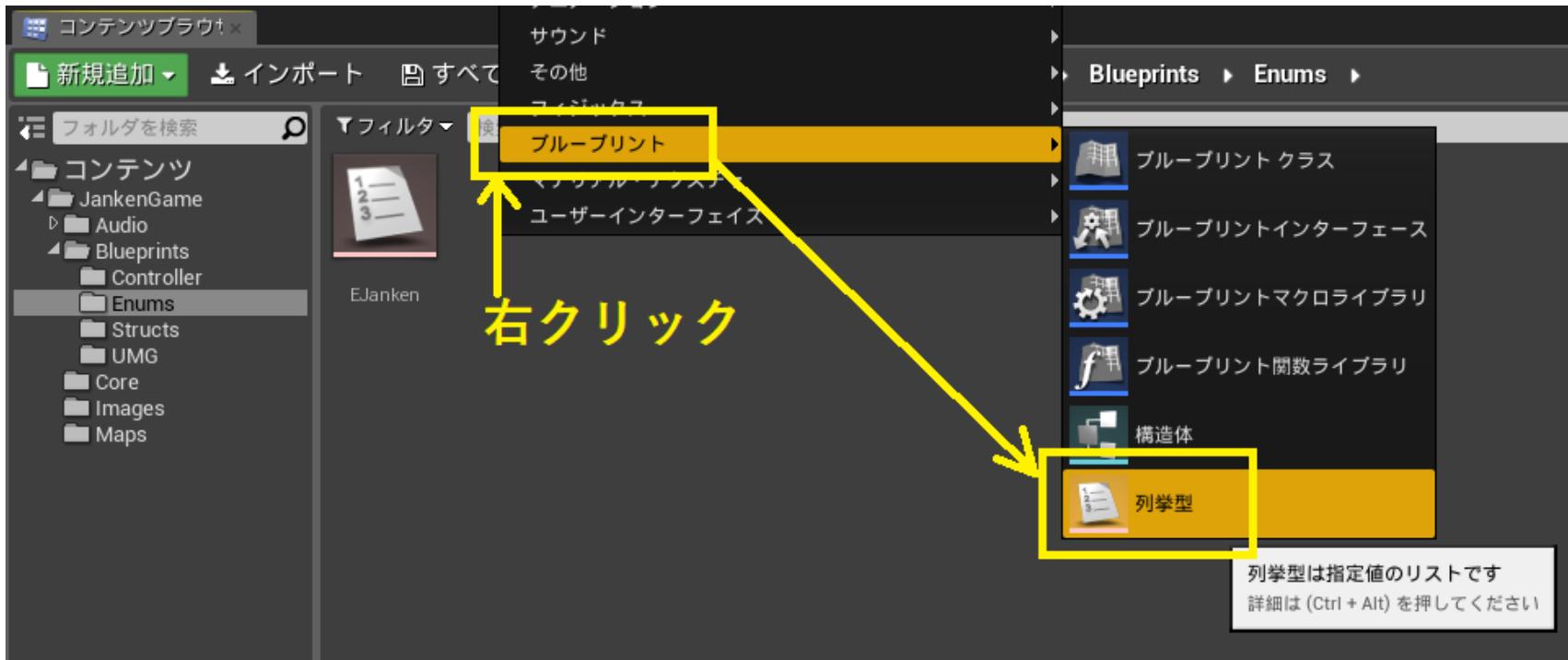
4.2 列挙型:EJankenResultを作成

4.3 関数:PlayEnemyJankenのアウトプットの型をEJankenに変更

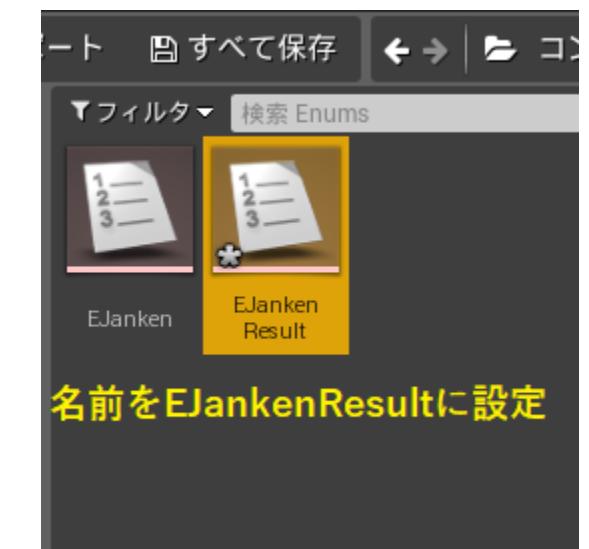
4.4 関数: JudgeJankenのインプットの型をEjanken,
アウトプットの型をEJankenResultに変更

4.5 関数のインプット・アウトプットの変更によるエラー修正

列挙型::EJankenResultを作成



右クリック > ブループリント > 列挙型



名前をEJankenResultに設定

エニュメレーターを設定 Description(説明)も設定



エニュメレーター

Display Name	Description
Win	勝ち
Lose	負け
Draw	あいこ

Description

Enum Description

じゃんけんの勝敗種別

4. 列挙型を使ってエラーが少ない処理にする

4.1 列挙型:EJankenを作成

4.2 列挙型:EJankenResultを作成

4.3 関数:PlayEnemyJankenのアウトプットの型をEJankenに変更

4.4 関数: JudgeJankenのインプットの型をEjanken,
アウトプットの型をEJankenResultに変更

4.5 関数のインプット・アウトプットの変更によるエラー修正

関数:PlayEnemyJankenのアウトプットの型をEJankenに変更



BP_JankenGameControllerを開く
PlayEnemyJankenを編集



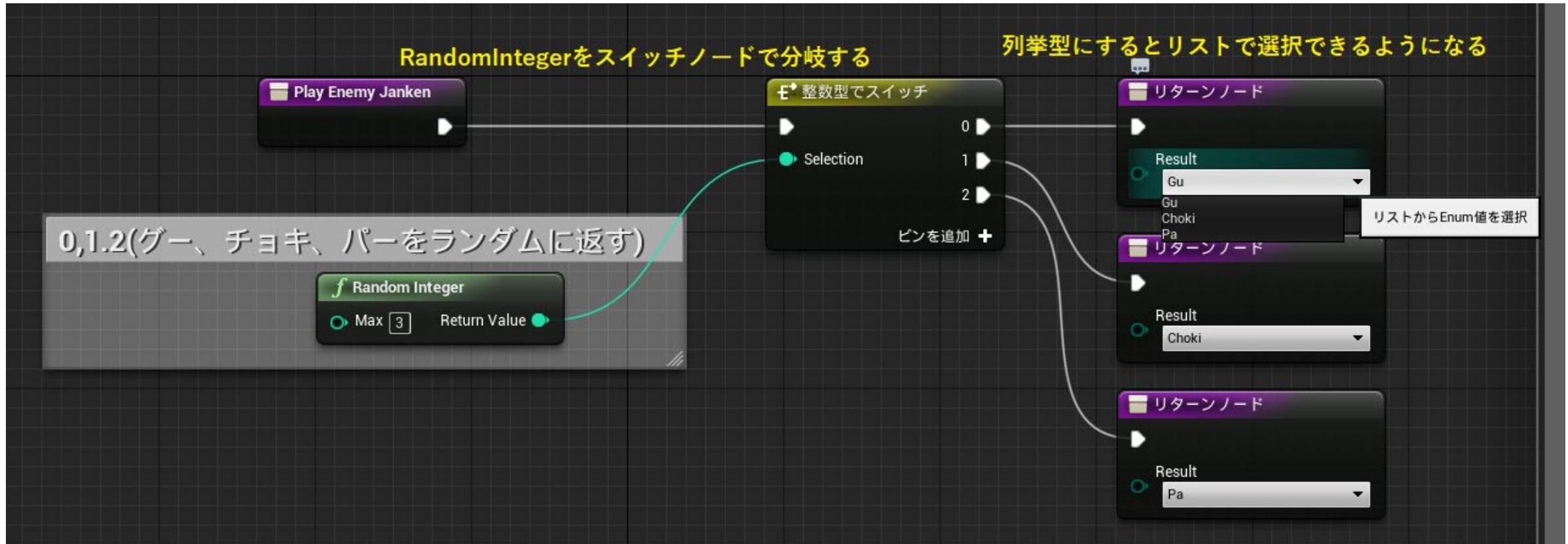
独自に作成したクラスや列挙型は検索しないと
見つけづらい

関数名	アクセス指定子	Input/Output	パラメータ名	型
PlayEnemyJanken	プライベート	Output	Result	Integer



関数名	アクセス指定子	Input/Output	パラメータ名	型
PlayEnemyJanken	プライベート	Output	Result	EJanken

RandomIntegerのReturnValueの値をスイッチノードで分岐し、リターンノードに値を設定する



RandomIntegerのReturnValueの値をスイッチノードで分岐し、リターンノードに値を設定する

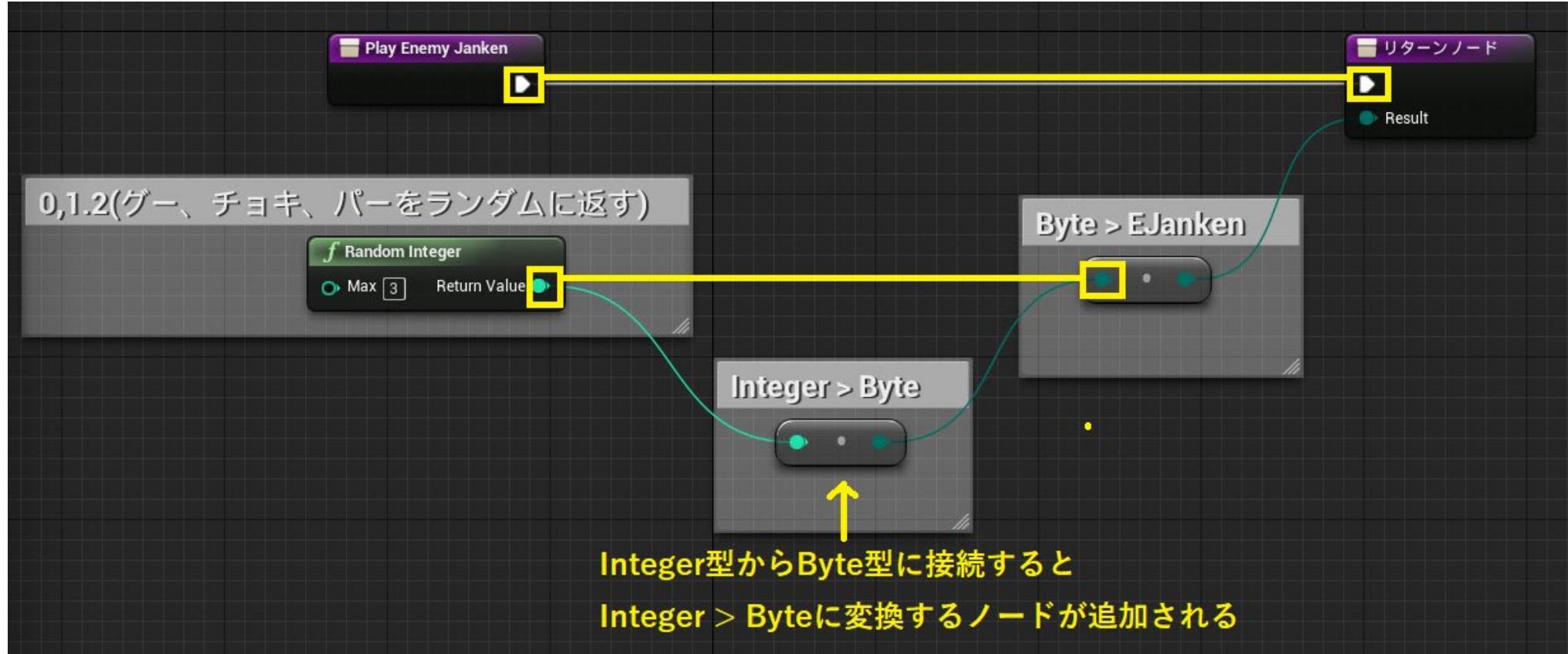
列挙型は数値(Byte)に変換することが出来る

EJanken	数値 (byte:0~255)
Gu	0
Choki	1
Pa	2

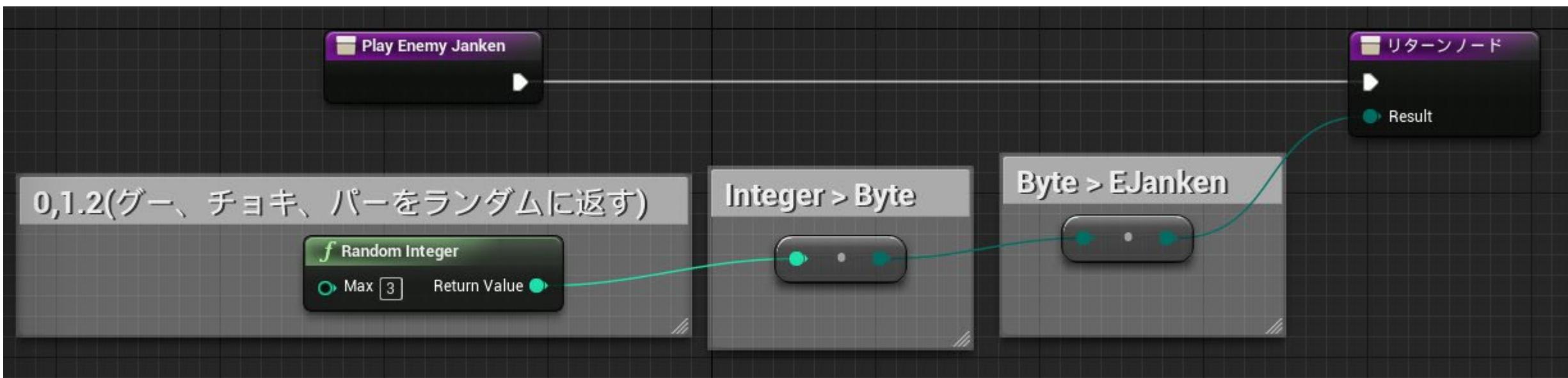
列挙型は数値に変換することが出来る
0から始まり、
エナミュレーターが追加される
毎にインクリメントされる



ランダムの数値からランダムの列挙型を作成することが出来る



列挙型Ejankenをランダムで作成する



4. 列挙型を使ってエラーが少ない処理にする

4.1 列挙型:EJankenを作成

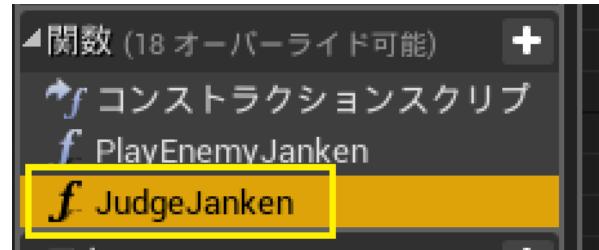
4.2 列挙型:EJankenResultを作成

4.3 関数:PlayEnemyJankenのアウトプットの型をEJankenに変更

4.4 関数: JudgeJankenのインプットの型をEjanken,
アウトプットの型をEJankenResultに変更

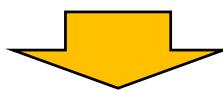
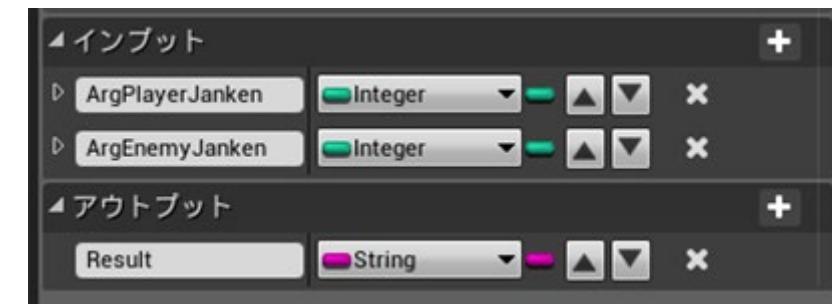
4.5 関数のインプット・アウトプットの変更によるエラー修正

関数: JudgeJankenの
インプットの型をEJanken,
アウトプットの型をEJankenResultに変更



JudgeJankenを選択する

関数名	アクセス指定子	Input/Output	パラメータ名	型
JudgeJanken	プライベート	Input	ArgPlayerJanken	Integer
		Input	ArgEnemyJanken	Integer
		Output	Result	String



インプット、アウトプットの型を列挙型に変更

関数名	アクセス指定子	Input/Output	パラメータ名	型
JudgeJanken	プライベート	Input	ArgPlayerJanken	EJanken
		Input	ArgEnemyJanken	EJanken
		Output	Result	EJankenResult



リターンノードに何を設定するか定義したマトリックスを
数値、文字列から列挙型のエニュメレーターに変更

リターンノードに設定する文字列

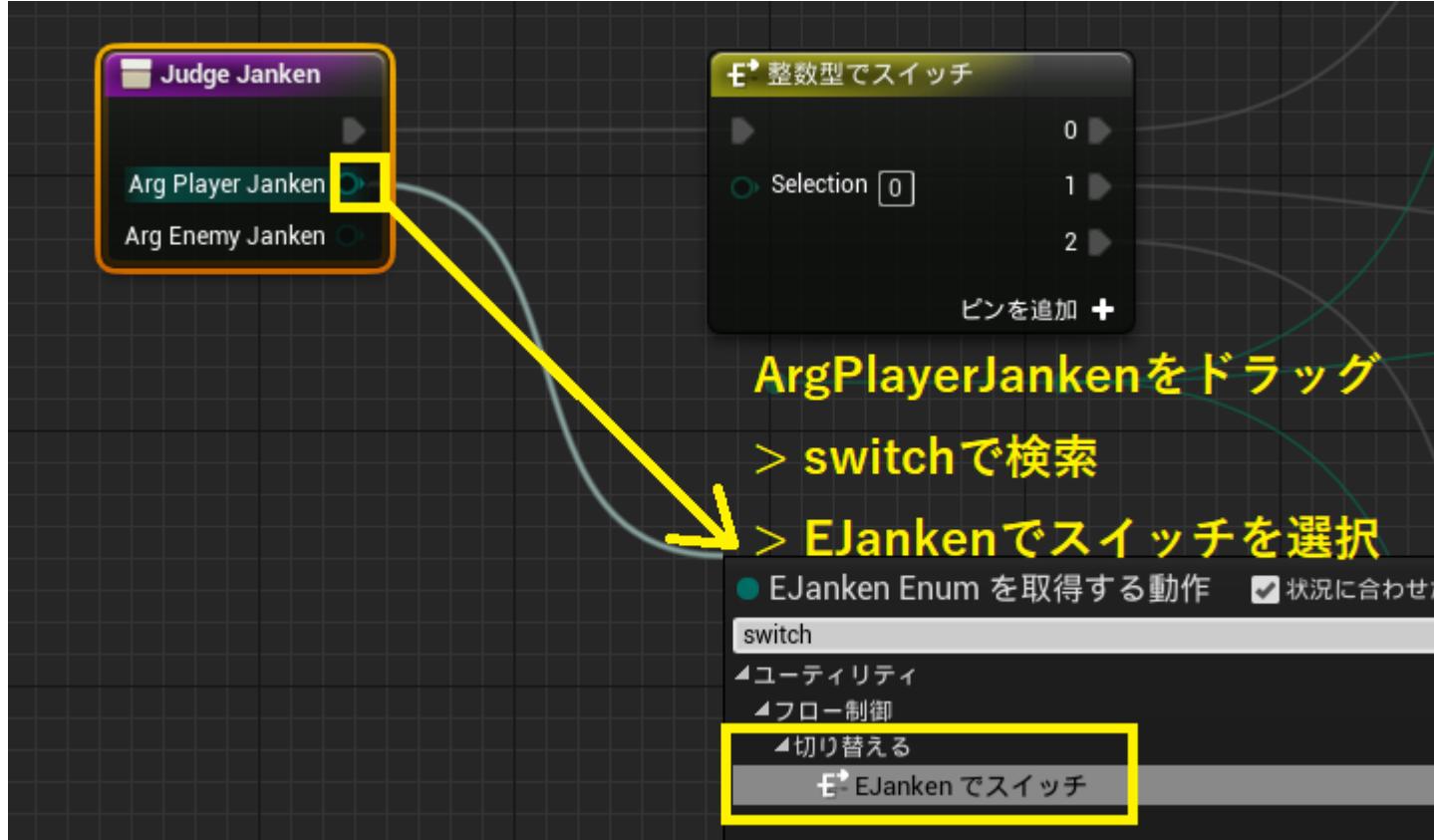
Player \ Enemy	グー(0)	チョキ(1)	パー(2)
グー(0)	Draw	Win	Lose
チョキ(1)	Lose	Draw	Win
パー(2)	Win	Lose	Draw

リターンノードに設定するエニュメレーター

Player \ Enemy	グー(Gu)	チョキ(Choki)	パー(Pa)
グー(Gu)	Draw	Win	Lose
チョキ(Choki)	Lose	Draw	Win
パー(Pa)	Win	Lose	Draw

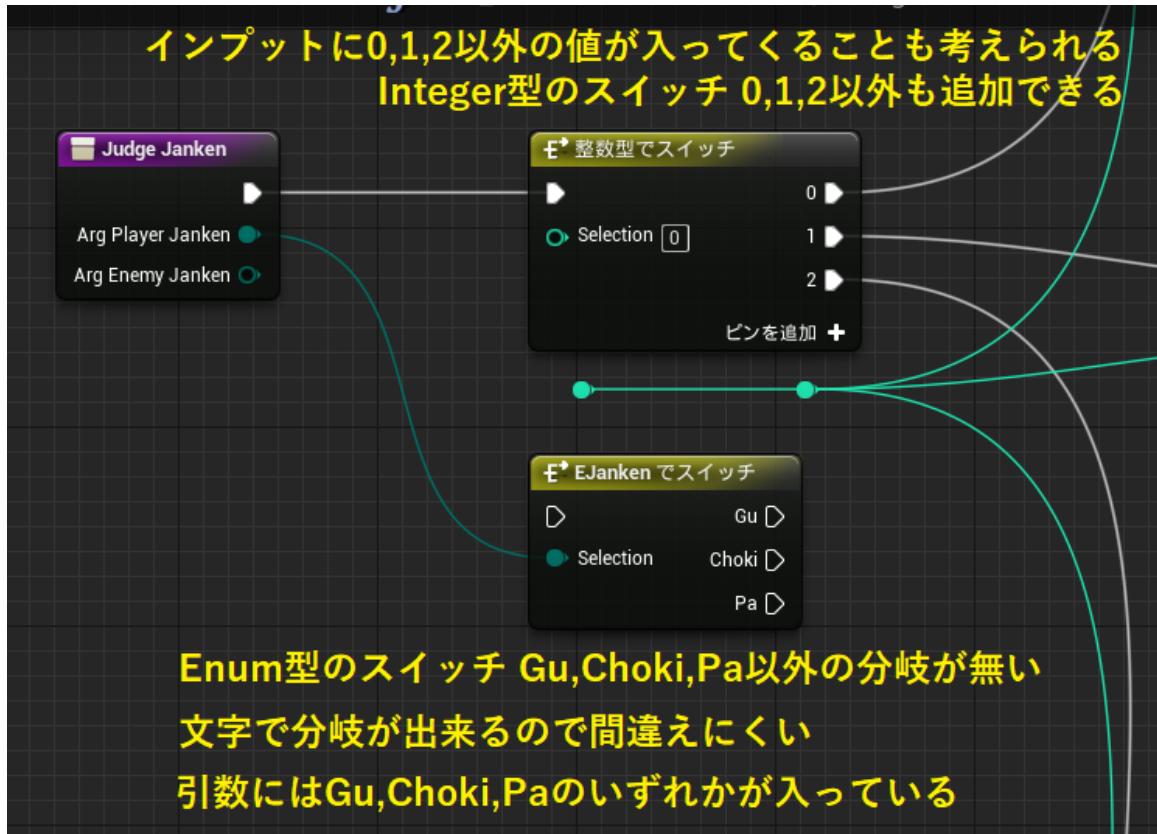


ArgPlayerJankenのスイッチノードを追加



ArgPlayerJankenをドラッグ
> switchで検索
> EJankenでスイッチ を選択

整数(Integer)型とEnum型の比較



	整数(Integer)型	Enum型
switch	ピンを追加 Default	エニュメレーター (Gu,Choki,Pa)の分岐
Input	意図していない値 が入ることがある	エニュメレーターに設定 した項目以外は入らない
Readable	0,1,2からじゃんけんの種 別を判断し づらい	文字からじゃんけんの種 別を判断できる



Enum型を使うことのメリット

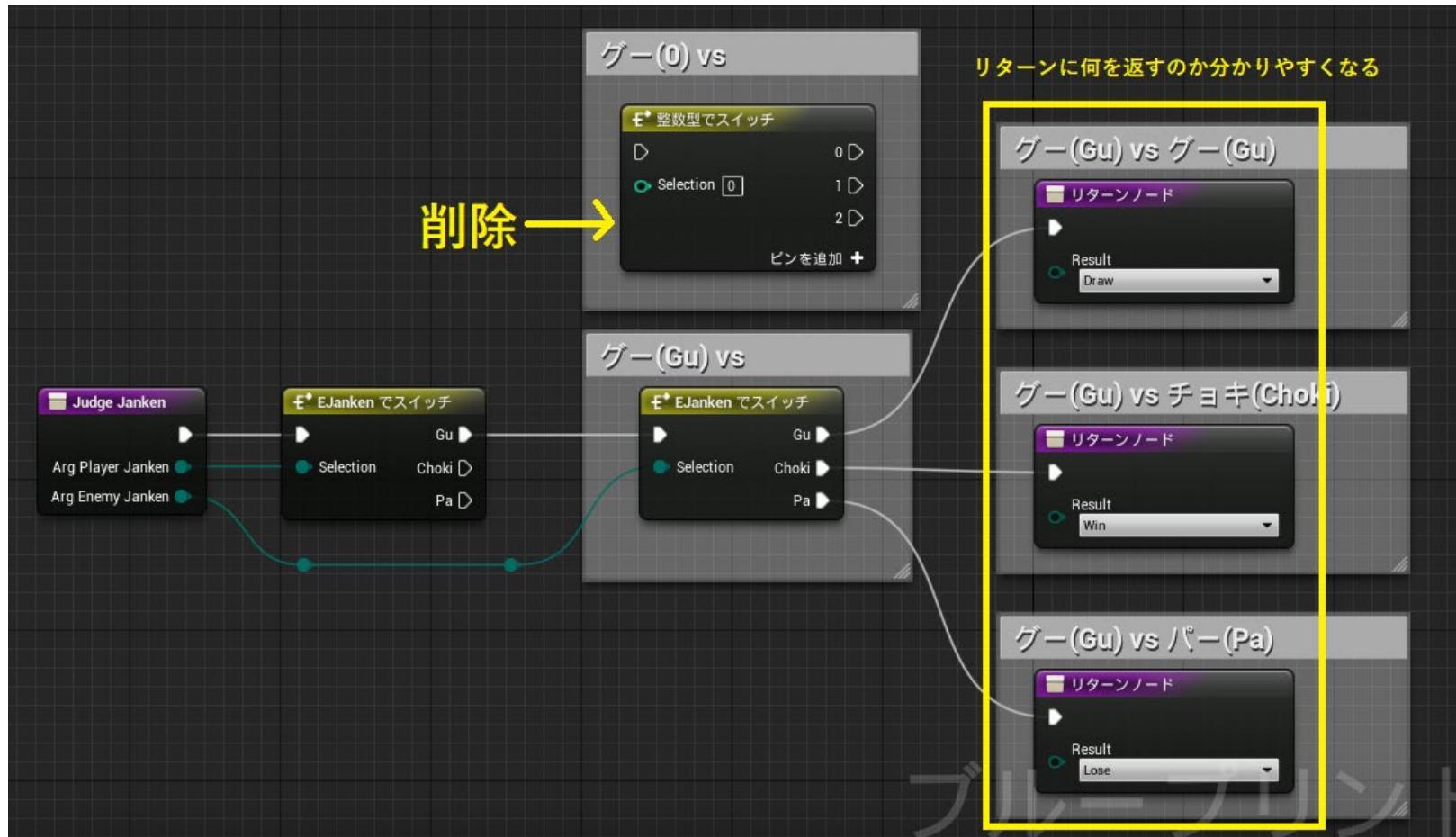
分岐を確定できる
意図した値しか入らない
グラフを読みやすくなる (可読性アップ)

ArgEnemyJankenのスイッチを追加



ArgEnemyJanken を ドラッグ
> switch で検索
> EJanken でスイッチ を 選択

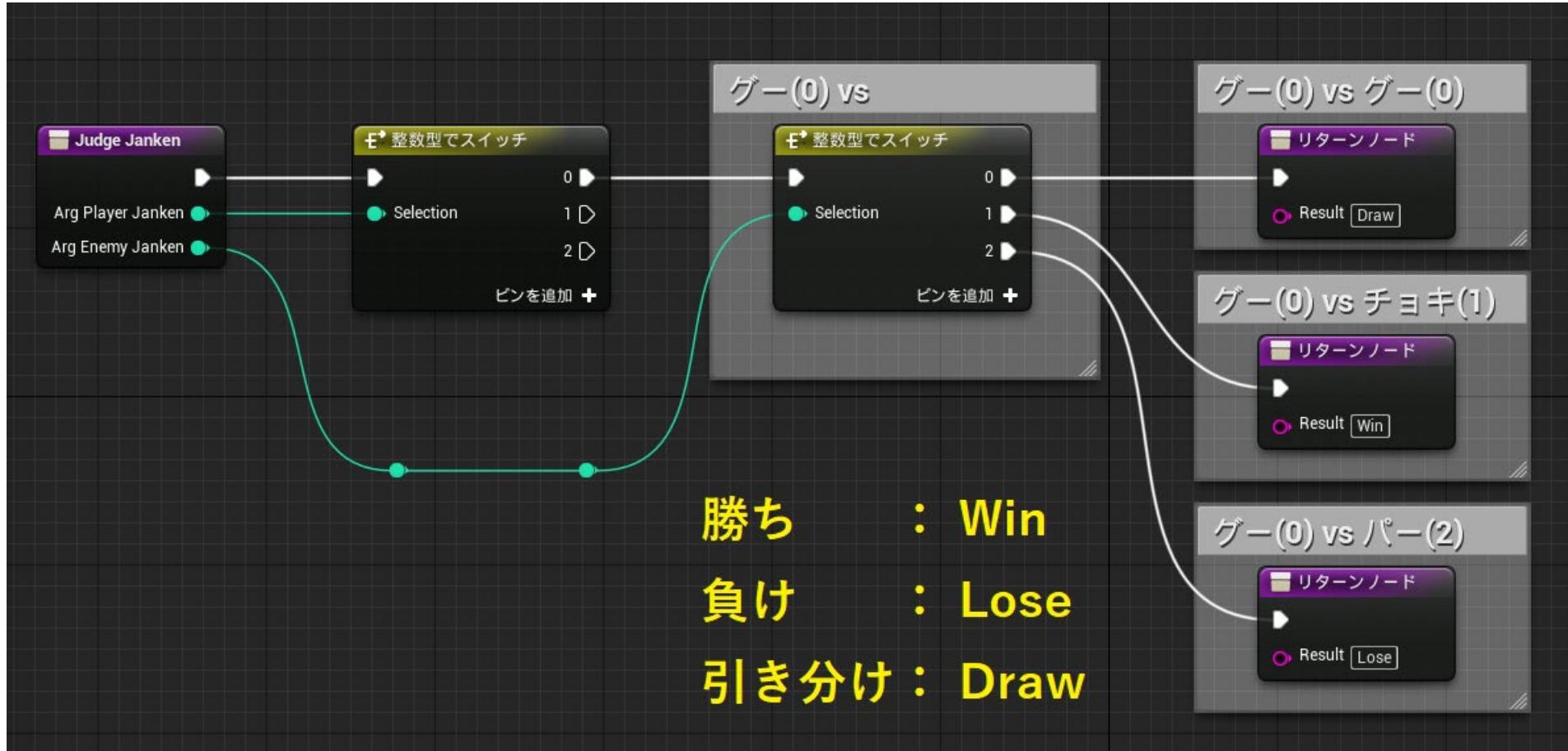
グー vs の分岐を行いリターンノードを設定する



Enemy Player	グー(Gu)	チョキ (Choki)	パー(Pa)
グー(Gu)	Draw	Win	Lose
チョキ (Choki)	Lose	Draw	Win
パー(Pa)	Win	Lose	Draw

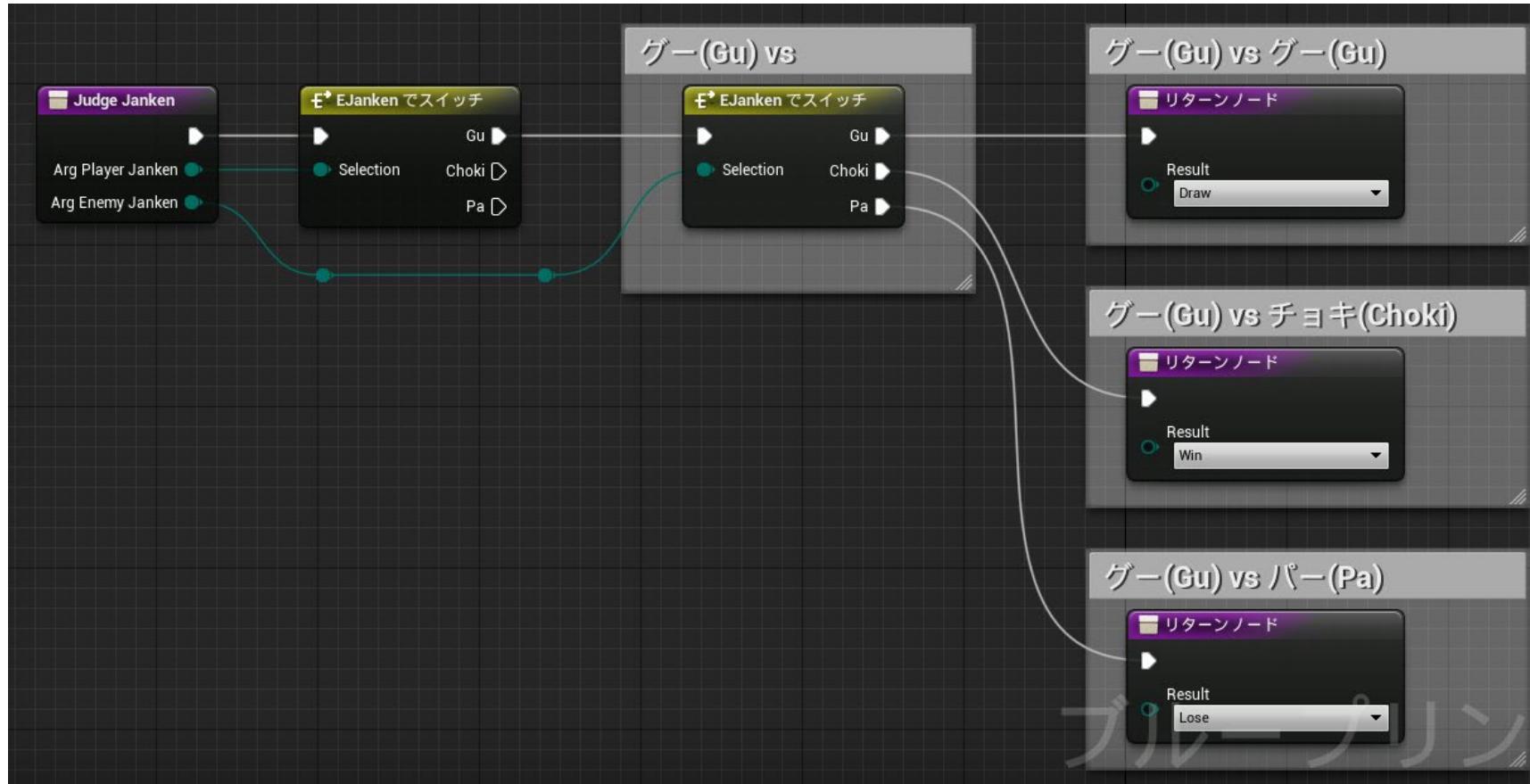
プレイヤーがグーの時の分岐からリターンノードを設定する
リターンノードはリストから選択なので、何を設定するのかが分かりやすくなる

インプットがInteger、リターンノードがStringの時の処理



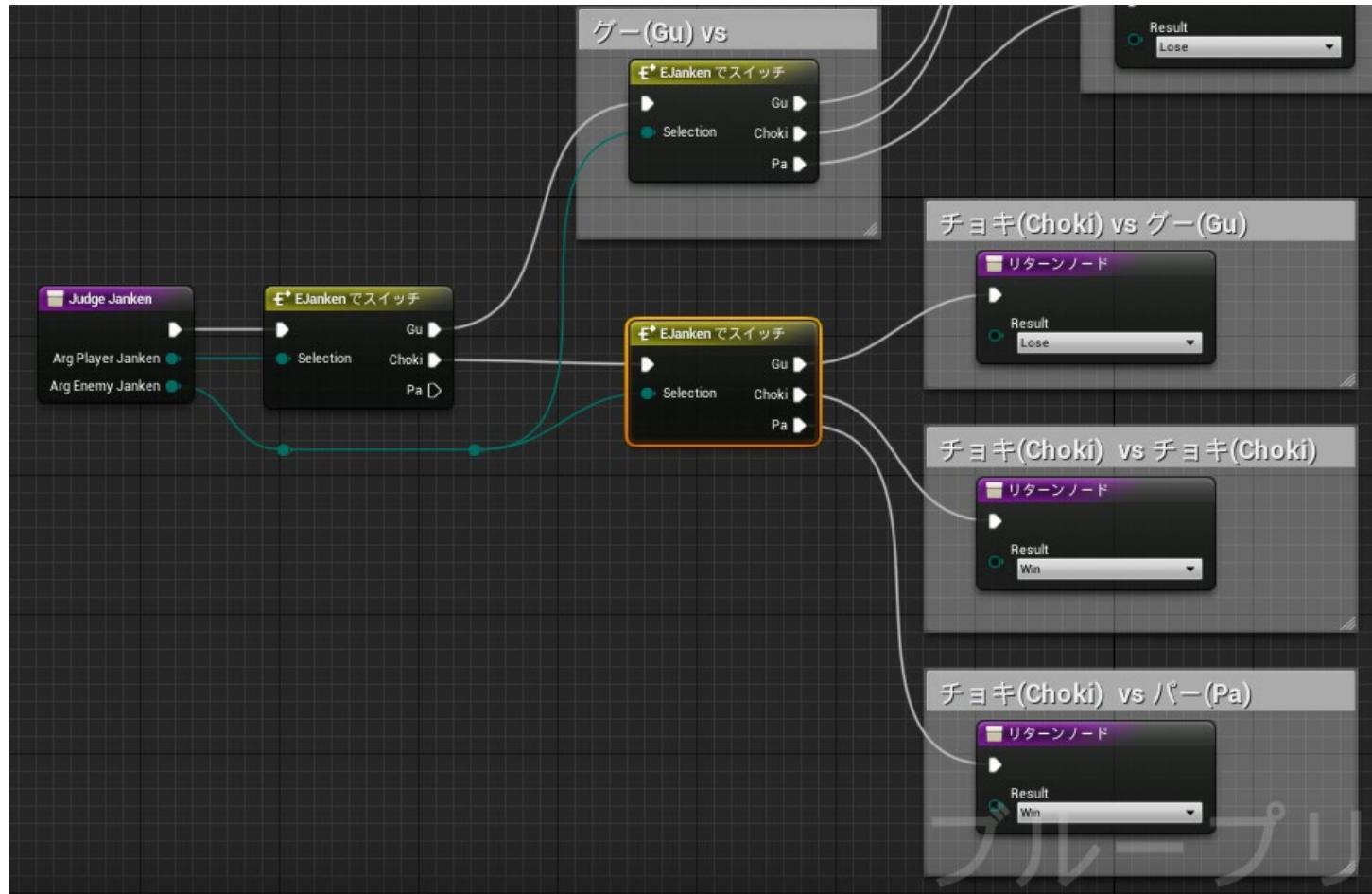
インプットがInteger、リターンノードがStringの時の処理
スイッチの数値は何で分岐しているのか分からない
リターンノードは手入力なので入力を間違える可能性がある

列挙型(Enum)を使用した処理



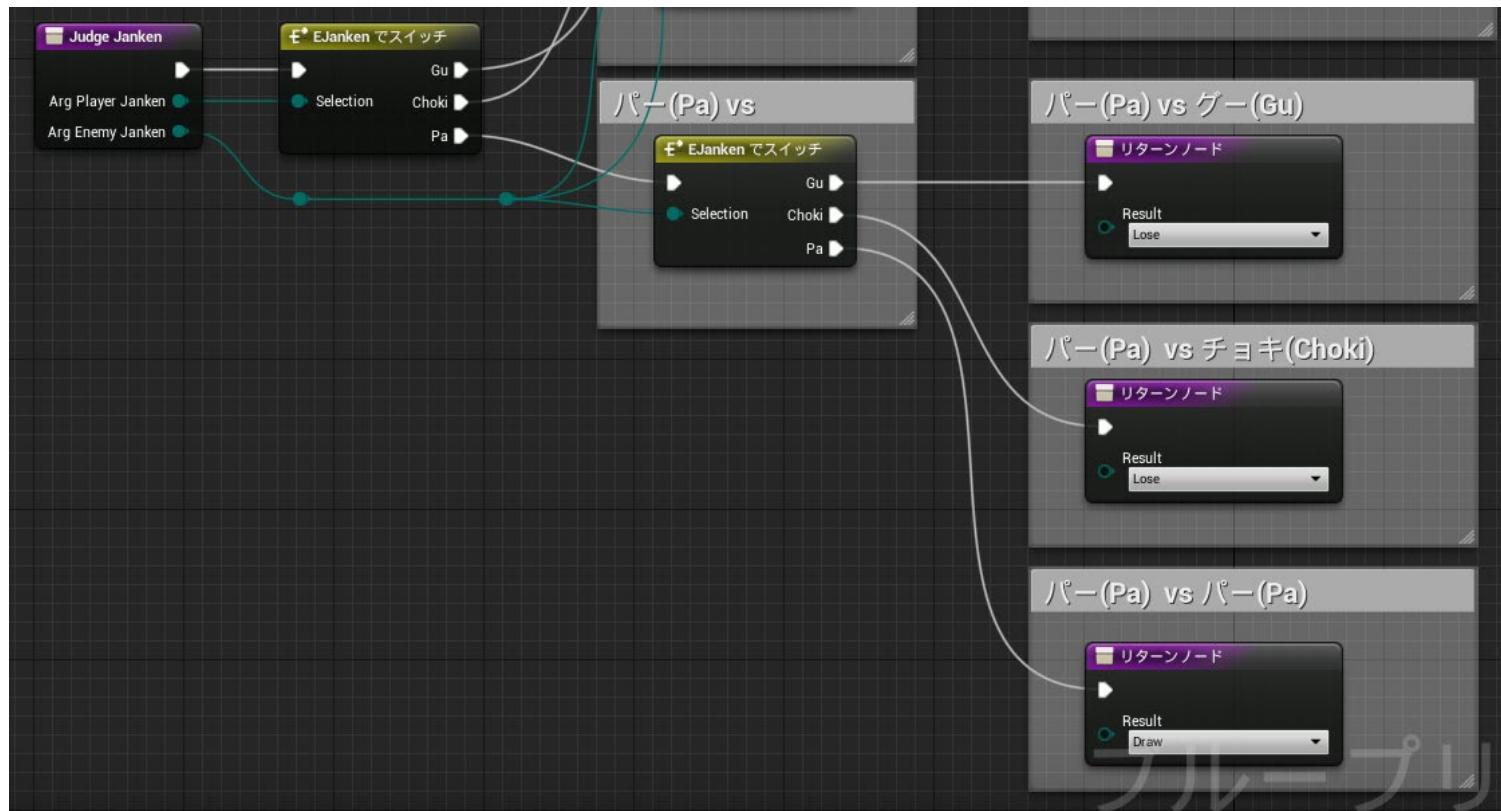
スイッチの分岐がエニュメレーターで設定した項目になるので、処理が分かりやすい
リターンノードはリストから選択なので、何を設定するのかが分かりやすくなる
間違えを少なくするために可読性を上げる

チョキ(Choki) vs の分岐を行いリターンノードを設定する



Player	Enemy	グー(Gu)	チョキ(Choki)	パー(Pa)
Player	グー(Gu)	Draw	Win	Lose
Enemy	チョキ(Choki)	Lose	Draw	Win
Player	パー(Pa)	Win	Lose	Draw

パー(Pa) vs の分岐を行いリターンノードを設定する

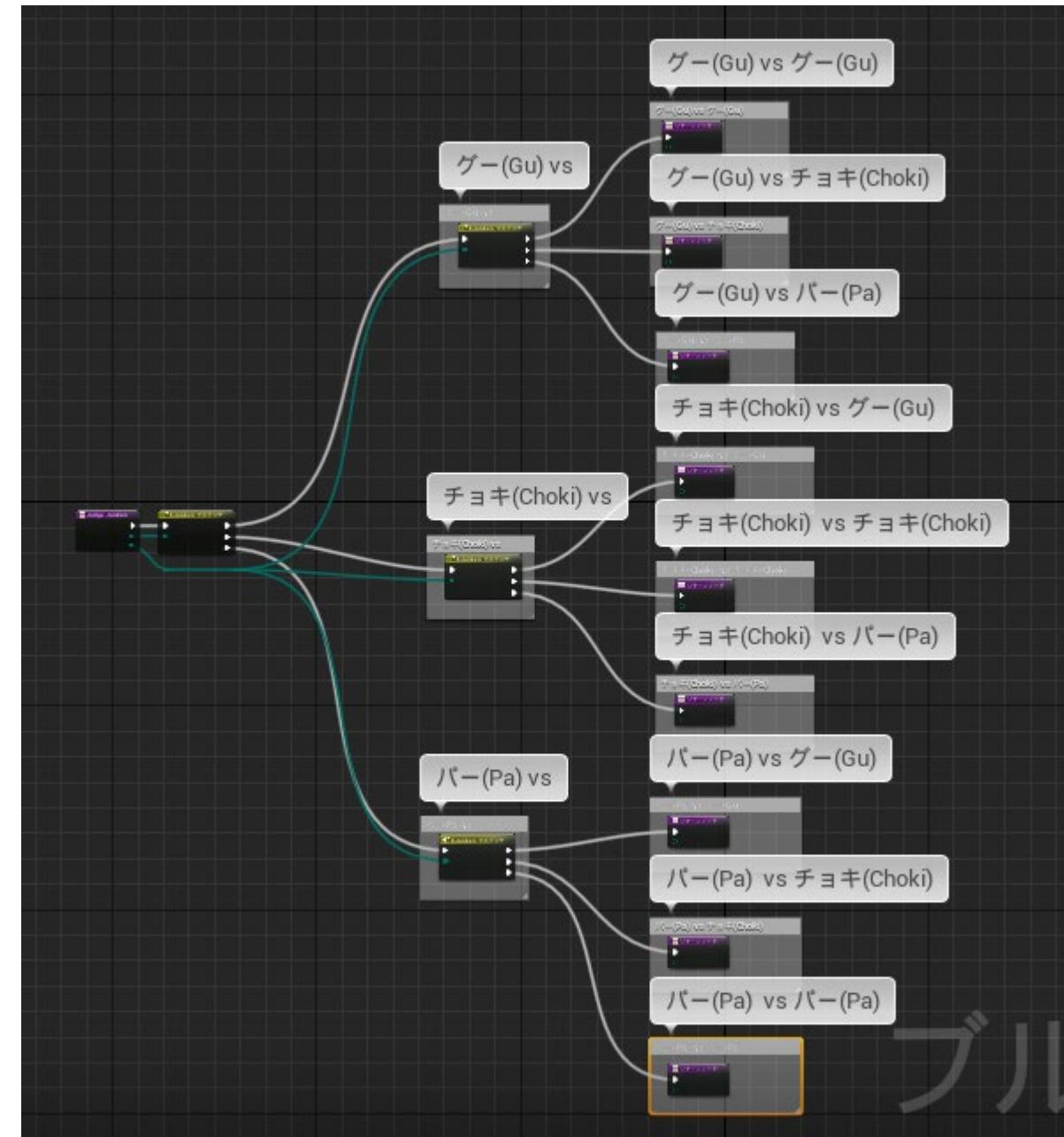


Player	Enemy	グー(Gu)	チョキ(Choki)	パー(Pa)
Player	グー(Gu)	Draw	Win	Lose
Player	チョキ(Choki)	Lose	Draw	Win
Player	パー(Pa)	Win	Lose	Draw

完成図

リターンノードに設定するエニュメレーター

Player	Enemy	グー(Gu)	チョキ(Choki)	パー(Par)
グー(Gu)	グー(Gu)	Draw	Win	Lose
チョキ(Choki)	チョキ(Choki)	Lose	Draw	Win
パー(Par)	パー(Par)	Win	Lose	Draw



4. 列挙型を使ってエラーが少ない処理にする

4.1 列挙型:EJankenを作成

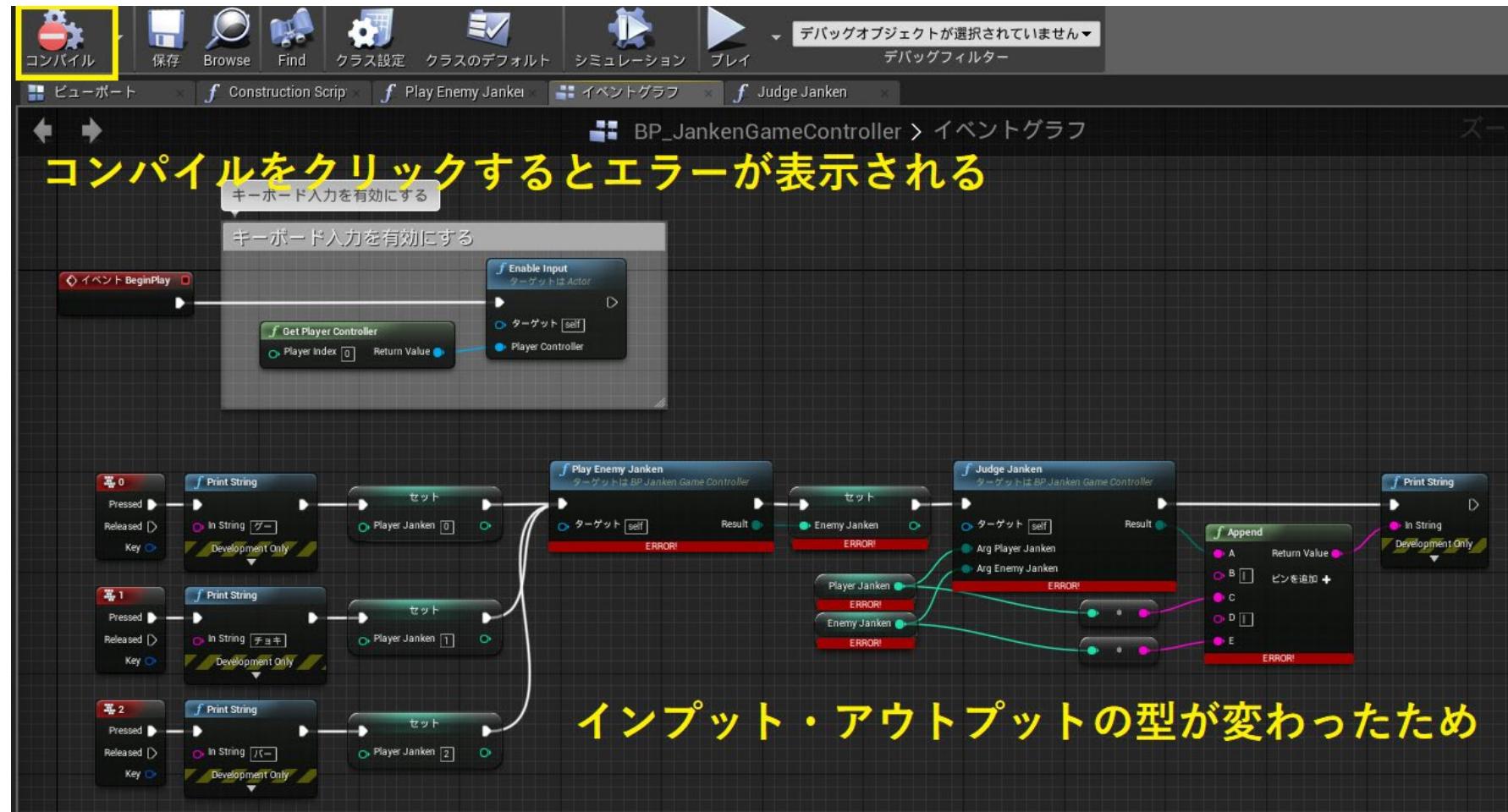
4.2 列挙型:EJankenResultを作成

4.3 関数:PlayEnemyJankenのアウトプットの型をEJankenに変更

4.4 関数: JudgeJankenのインプットの型をEjanken,
アウトプットの型をEJankenResultに変更

4.5 関数のインプット・アウトプットの変更によるエラー修正

コンパイルを行いエラーを表示する



変数の型を変更



変数名	変数の型	デフォルト値
PlayerJanken	Integer	-(設定なし)
EnemyJanken	Integer	-(設定なし)



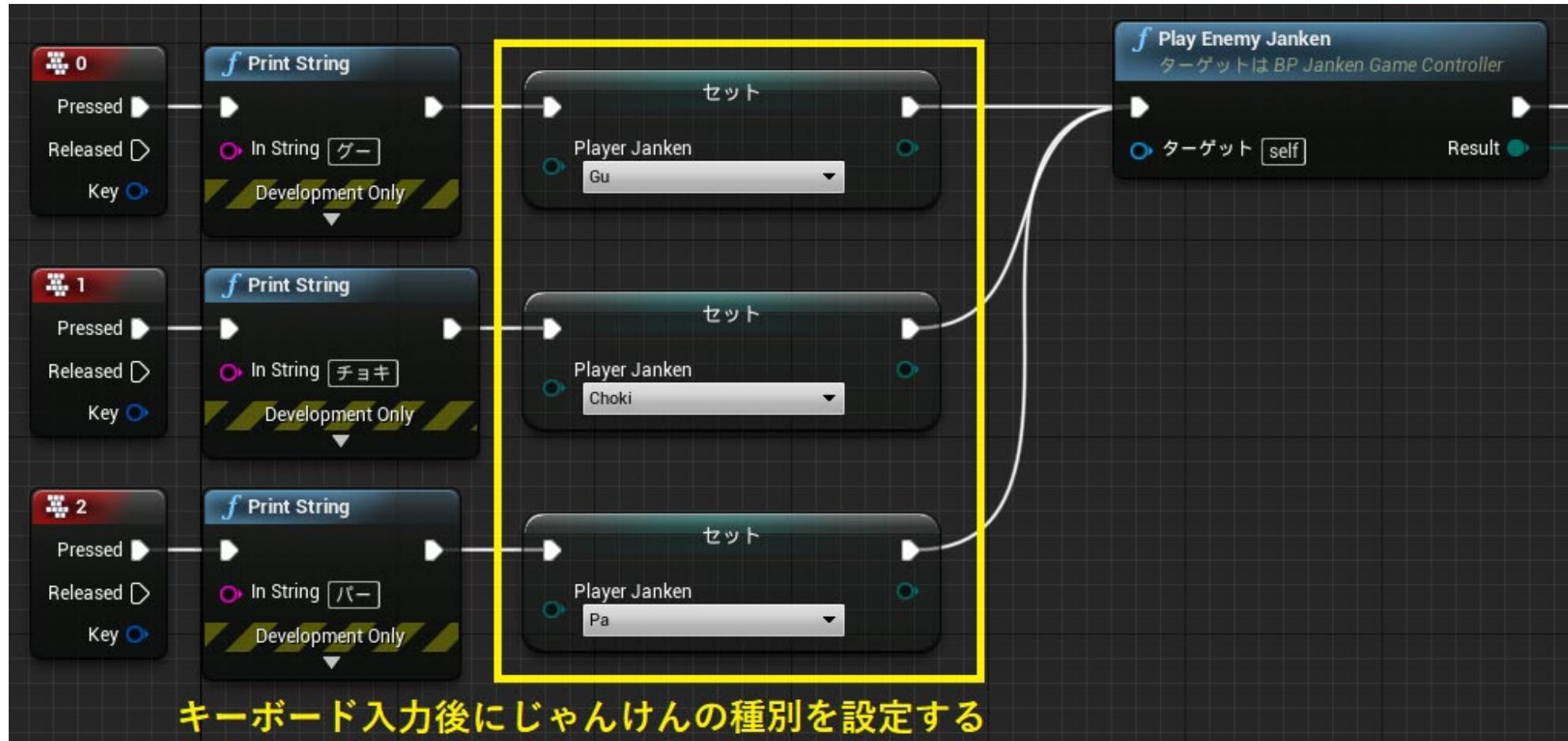
変数名	変数の型	デフォルト値
PlayerJanken	EJanken	-(設定なし)
EnemyJanken	EJanken	-(設定なし)

変数の型をIntegerからEJankenに変更する

変数の型をIntegerからEJankenに変更すると
ダイアログが表示される
> 変数の型を変更をクリック

エラー修正 1

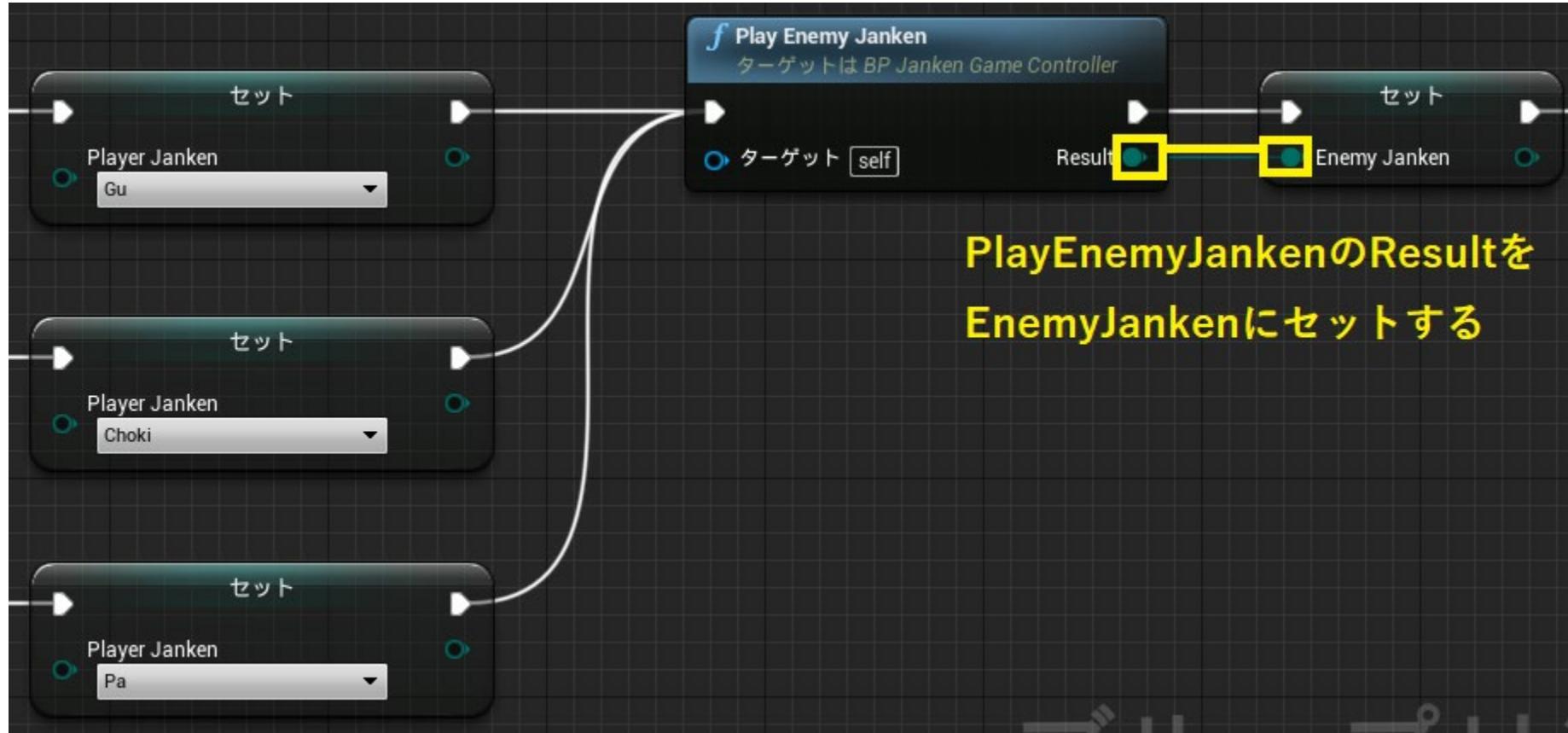
キーボード入力後にじゃんけんの種別をPlayerJankenにセットする



キーボード入力後にじゃんけんの種別をPlayerJankenに設定する

エラー修正 2

PlayEnemyJankenのResultをEnemyJankenにセットする



PlayerEnemyJankenのResultをEnemyJankenにセットする

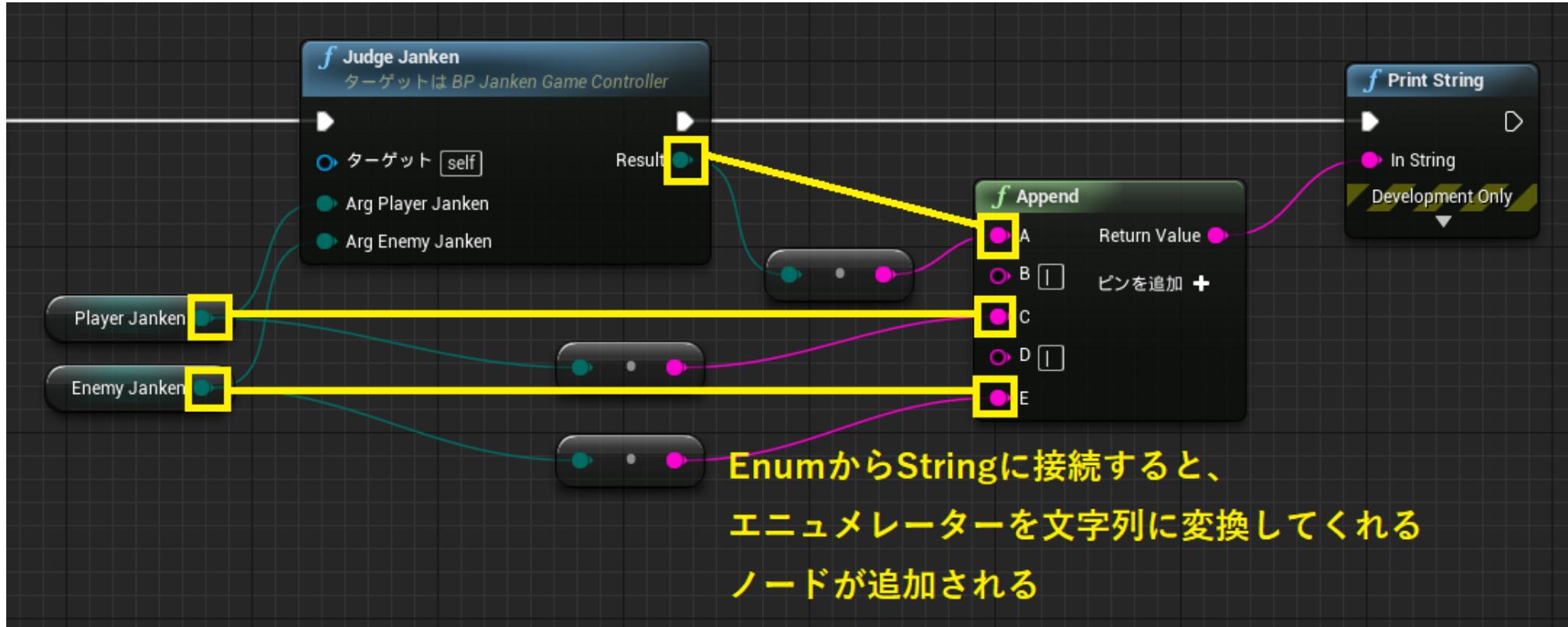
エラー修正 3-1

JudgeJankenのインプットを設定



エラー修正 3-2

AppendにEnumの値を設定する



Appendのインプットに列挙型を接続する

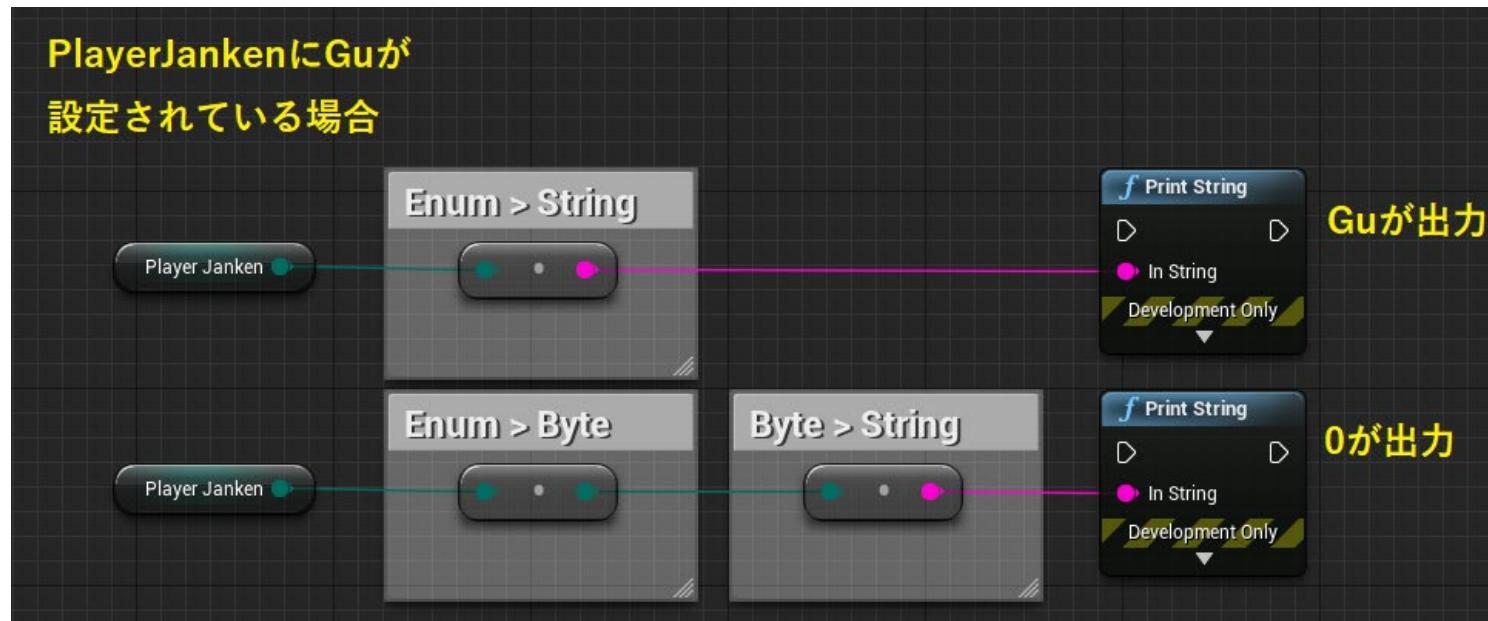
A: JudgeJankenのResult

C: PlayerJanken

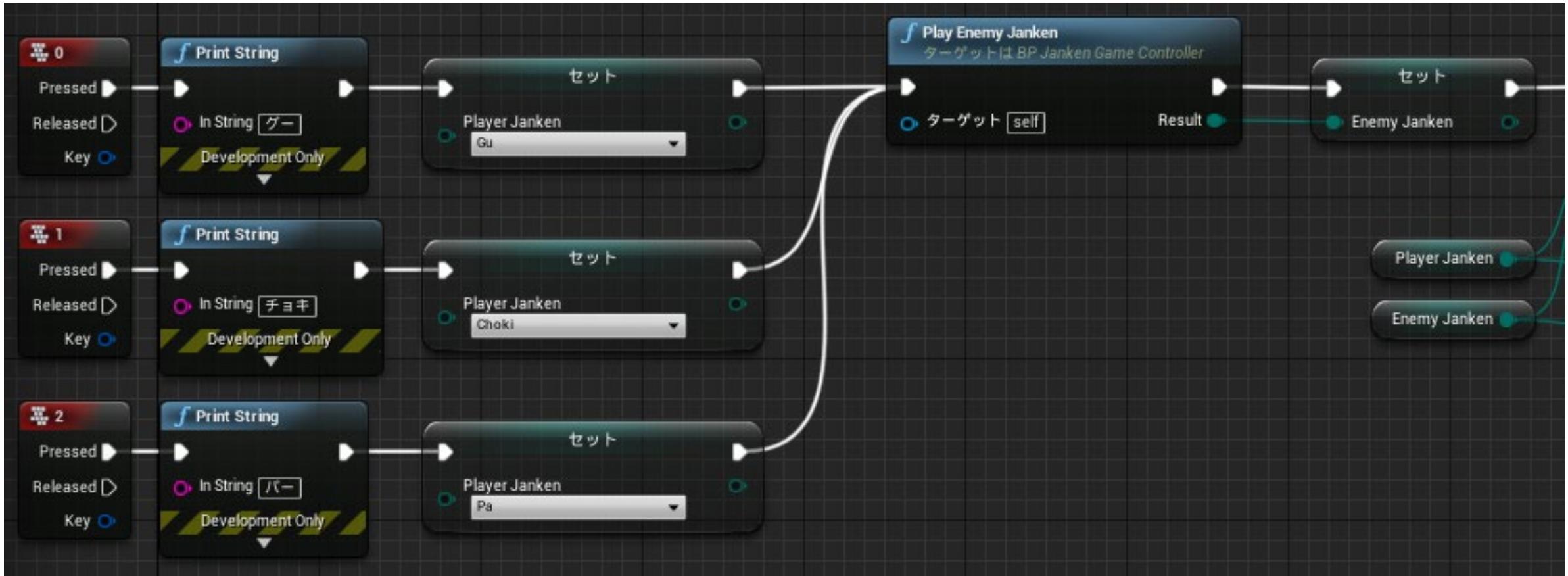
E: EnemyJanken

列挙型は0からの連番数値、文字列に変換できる

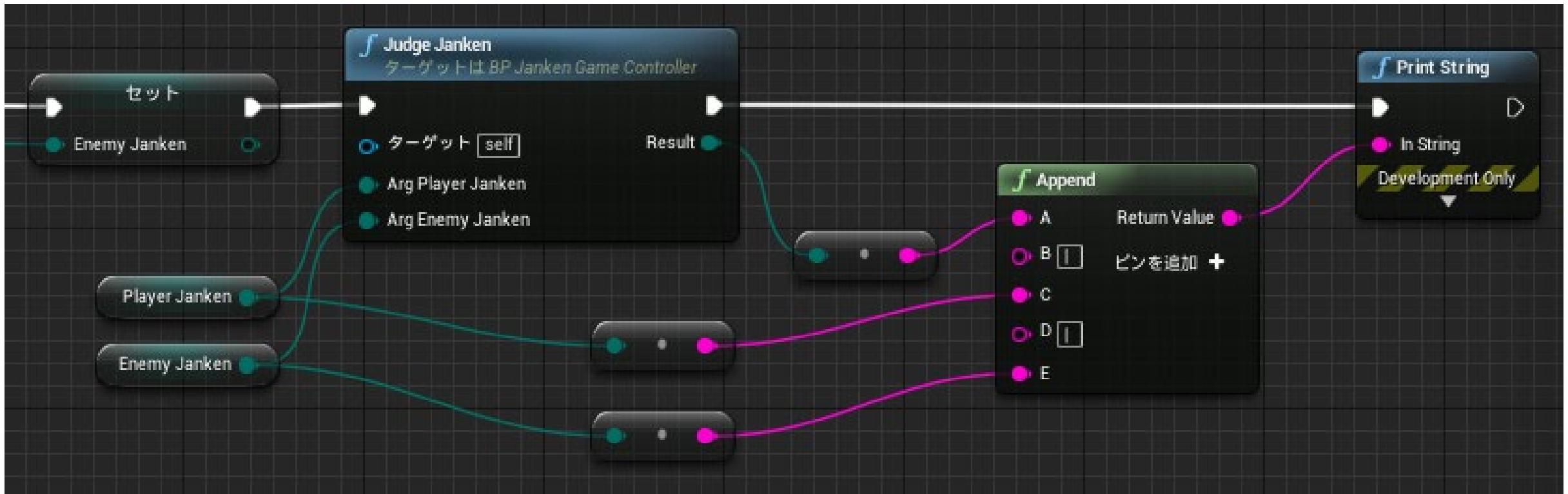
EJanken	数値 (byte:0~255)	文字列
Gu	0	Gu
Choki	1	Choki
Pa	2	Pa



エラー修正完了(左側)



エラー修正完了(右側)



プレイして確認する



Lose|Choki|Gu
チョキ
Draw|Gu|Gu
グー¹
Lose|Pa|Choki
パー²
Win|Choki|Choki
チョキ

5. 配列を使って5回勝負の結果を出力する

5. 配列を使って5回勝負の結果を出力する

5.1 フローを5回勝負に変更する

5.2 変数の追加(配列の変数)

5.3 配列の操作

5.4 フローの実装

5.5 カスタムイベントを作成して処理を切り離す

5. 配列を使って5回勝負の結果を出力する

5.1 フローを5回勝負に変更する

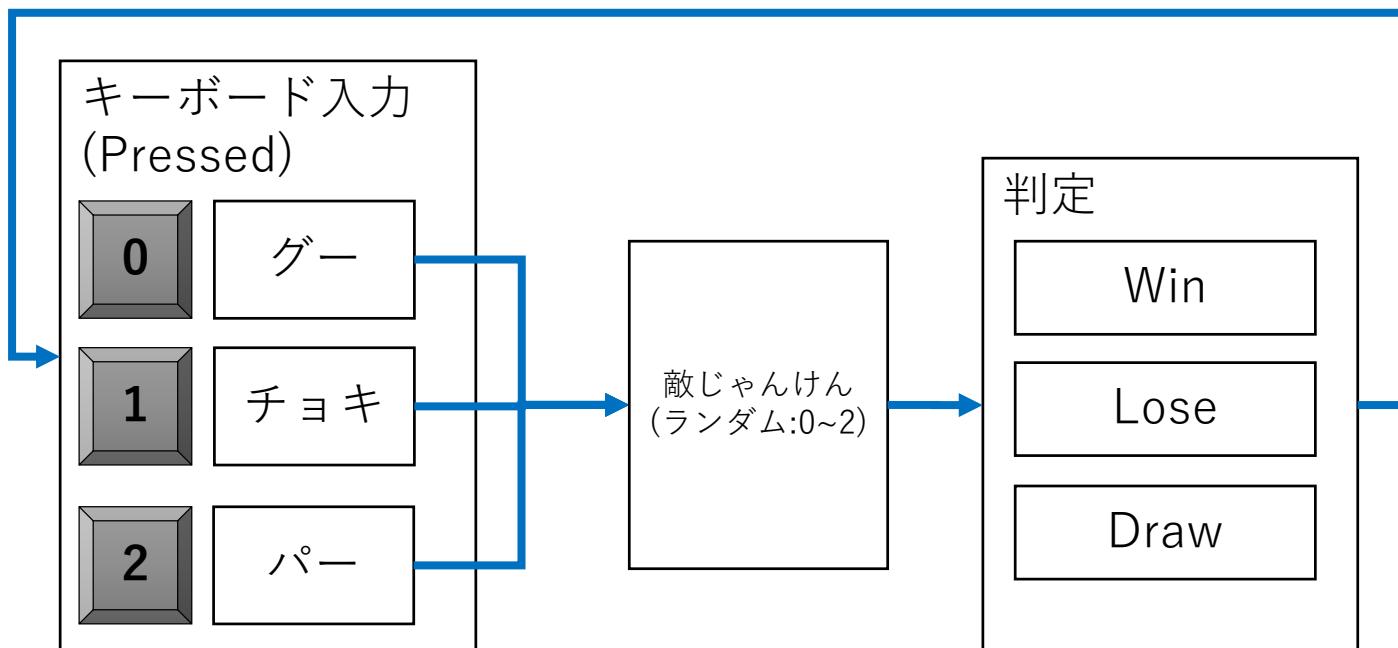
5.2 変数の追加(配列の変数)

5.3 配列の操作

5.4 フローの実装

5.5 カスタムイベントを作成して処理を切り離す

現状のフロー

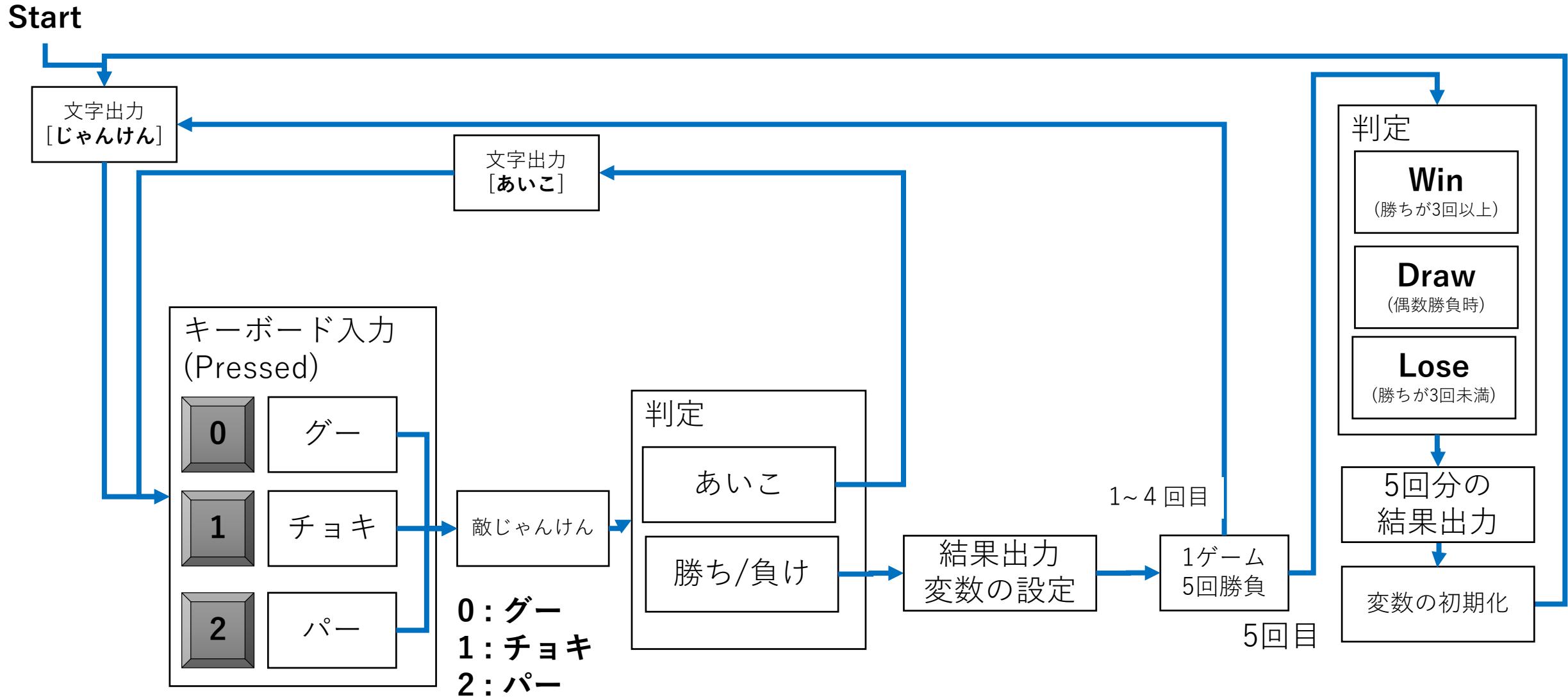


0: ゲー

1: チョキ

2: パー

5回勝負にフローを変更する



5. 配列を使って5回勝負の結果を出力する

5.1 フローを5回勝負に変更する

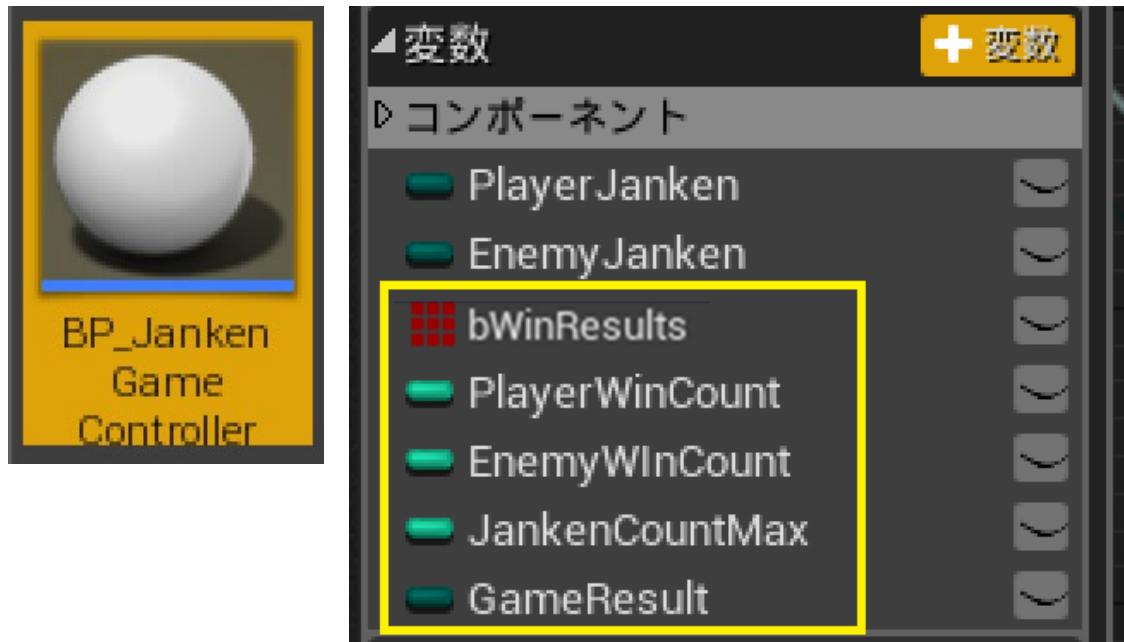
5.2 変数の追加(配列の変数)

5.3 配列の操作

5.4 フローの実装

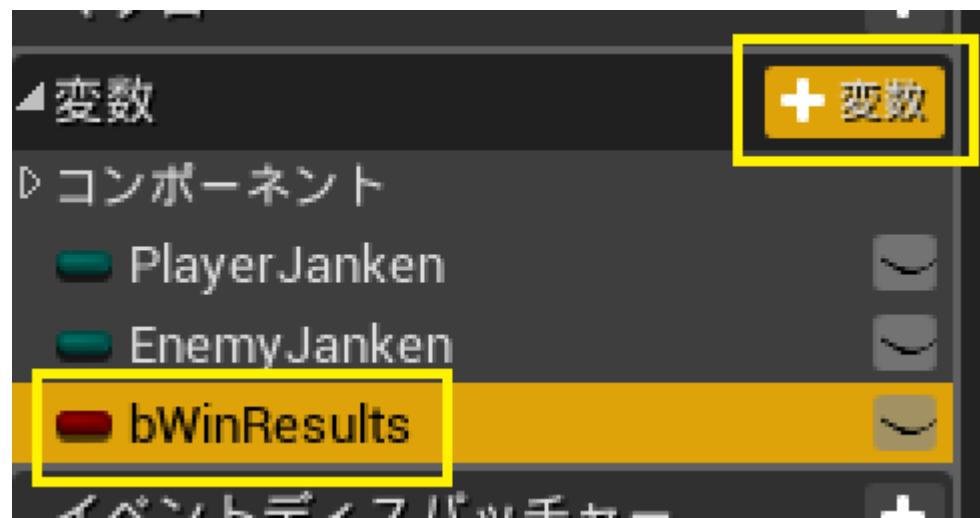
5.5 カスタムイベントを作成して処理を切り離す

変数の追加



変数名	変数の型	デフォルト値
PlayerJanken	EJanken	-(設定なし)
EnemyJanken	EJanken	-(設定なし)
bWinResults	Boolean (配列)	-(設定なし)
PlayerWinCount	Integer	-(設定なし)
EnemyWinCount	Integer	-(設定なし)
JankenCountMax	Integer	5
GameResult	EJankenResult	-(設定なし)

配列の変数の作成方法

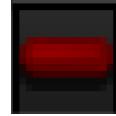


変数 bWinResultsを作成する
配列の場合は変数名を複数形にする



変数の型 Booleanの右側のアイコンをクリック
配列のアイコンをクリックすると配列に設定される

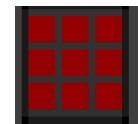
Single Variableと配列の違い



Single Variable

変数	値

Single Variableは値を1つしか持てない



配列

Index[0]	値
Index[1]	値
Index[2]	値
Index[3]	値
Index[4]	値

配列は同じ型の変数を複数持つことが出来る
Index（配列の番号）は0から始まる

5. 配列を使って5回勝負の結果を出力する

5.1 フローを5回勝負に変更する

5.2 変数の追加(配列の変数)

5.3 配列の操作

5.4 フローの実装

5.5 カスタムイベントを作成して処理を切り離す

配列の指定したIndex番号の値を取得したい場合



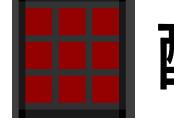
配列

Index[0]	値
Index[1]	値
Index[2]	値
Index[3]	値
Index[4]	値

Index番号2を
取得したい場合



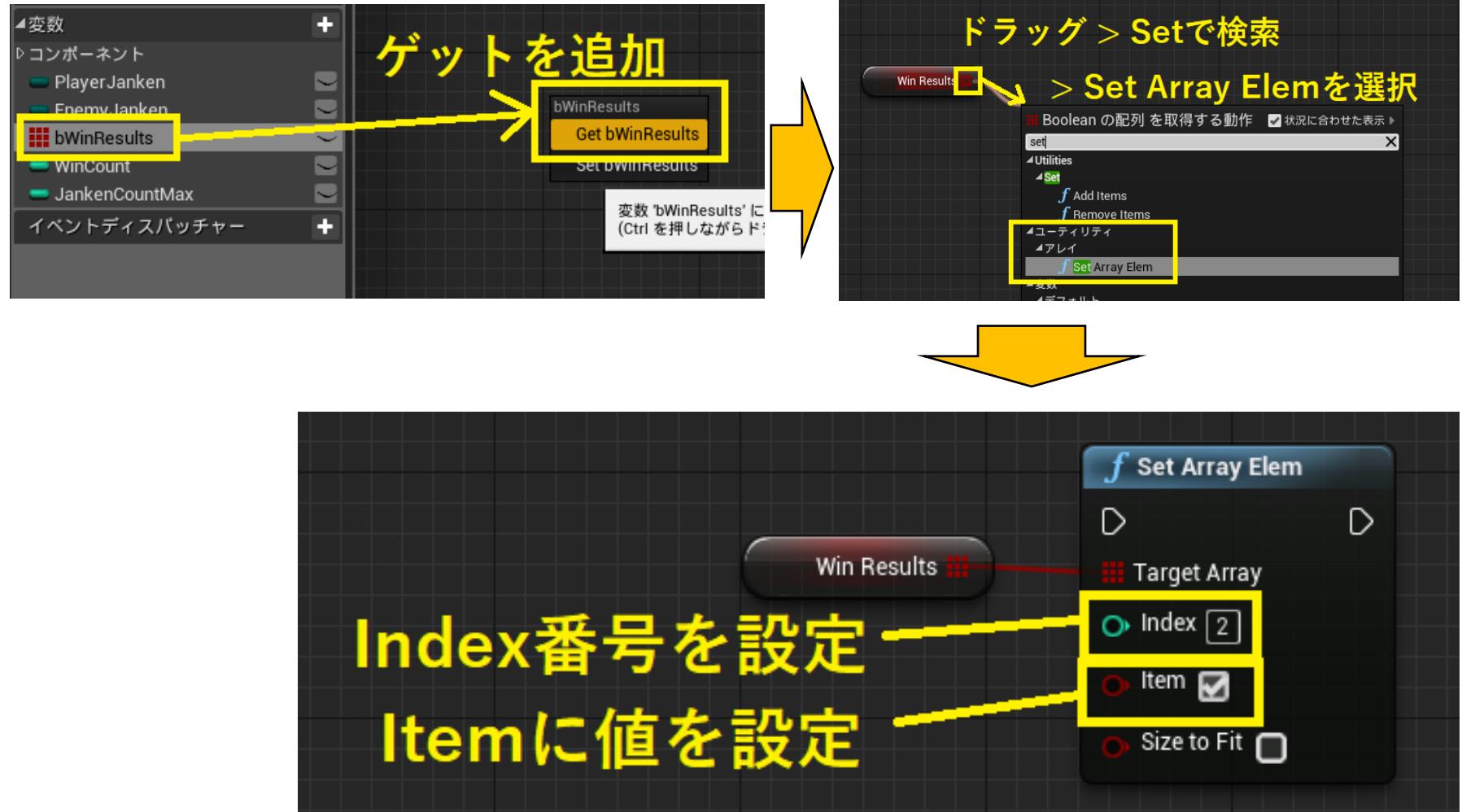
配列の指定したIndex番号の値を取得したい場合



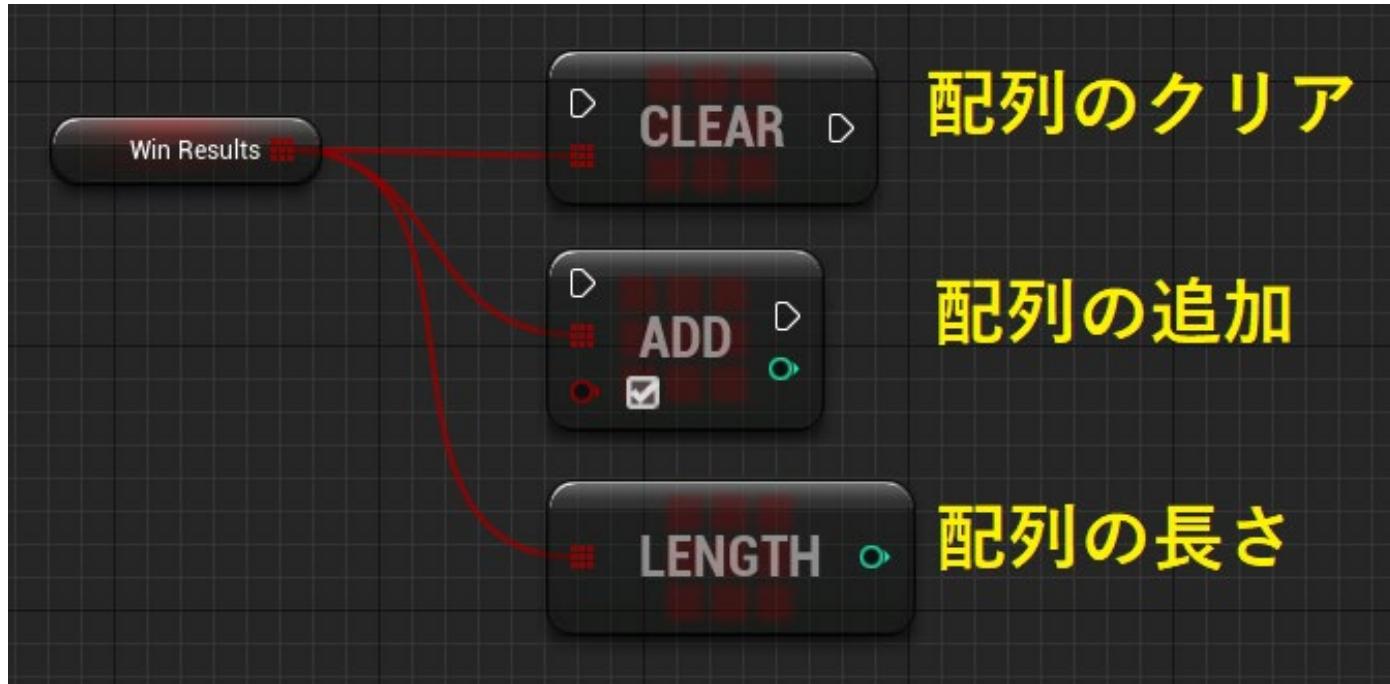
配列

Index[0]	値
Index[1]	値
Index[2]	値
Index[3]	値
Index[4]	値

Index番号2を
設定したい場合



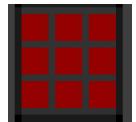
よく使用する配列を操作する関数



関数名	動き
Clear	配列を空にする
Add	配列を末尾に追加する
Length	配列数を取得する

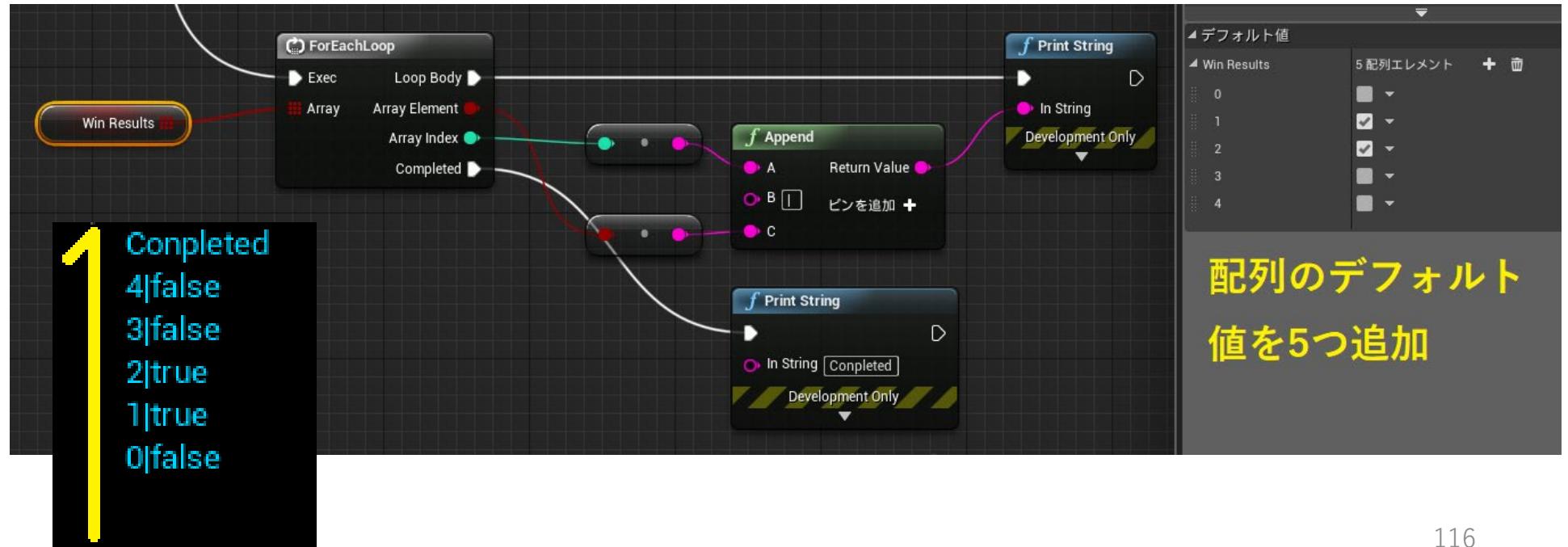
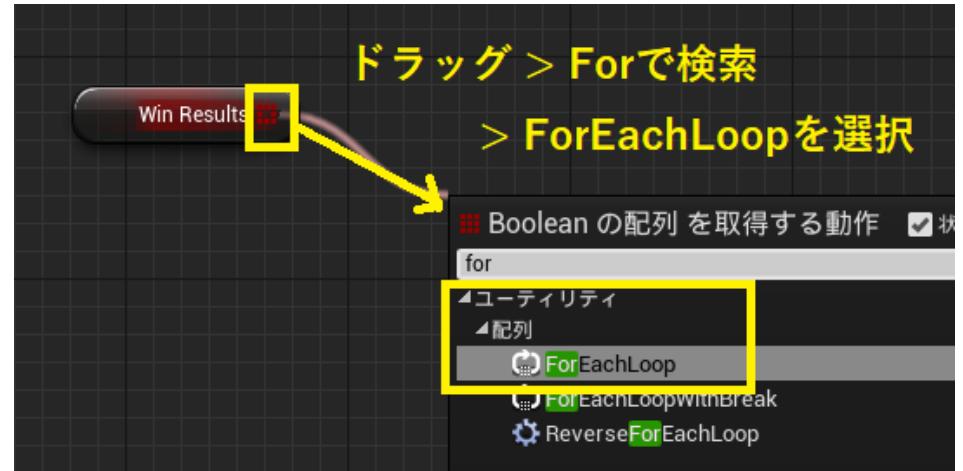
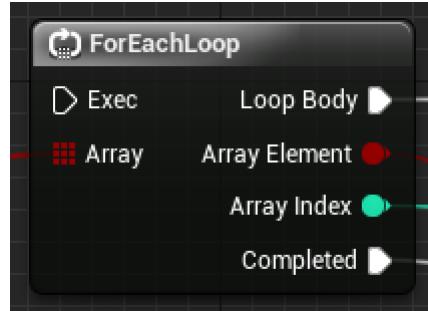
Clear,Add,Lengthはよく使用する

ForEachLoopを使用して配列を
0からラストまで取得する



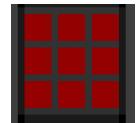
配列

Index[0]	値
Index[1]	値
Index[2]	値
Index[3]	値
Index[4]	値



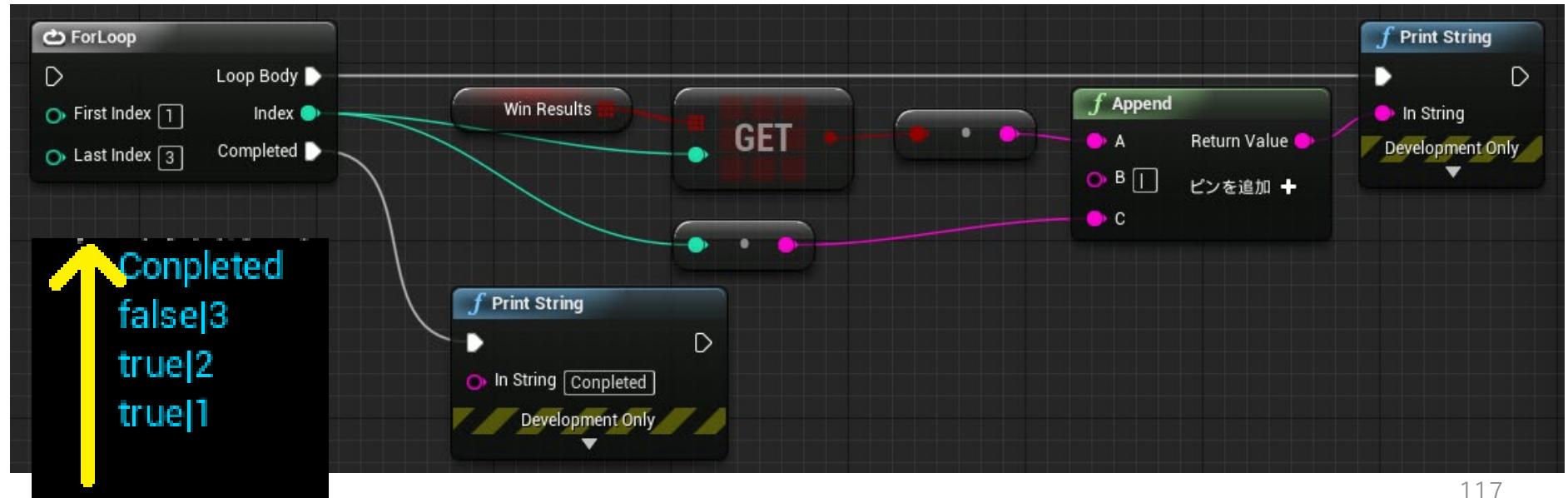
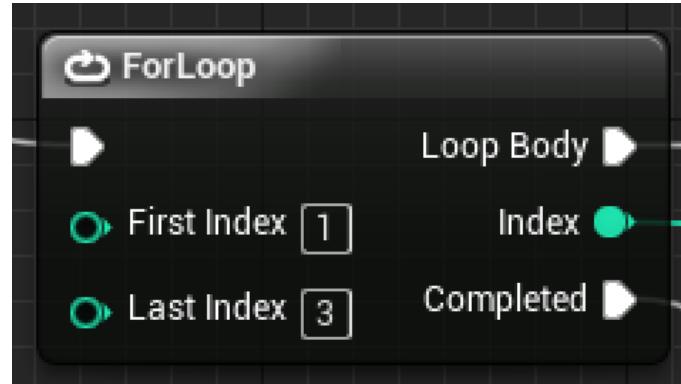
配列のデフォルト
値を5つ追加

ForLoopを指定すれば開始と終了のIndexを指定できる



配列

Index[0]	値
Index[1]	値
Index[2]	値
Index[3]	値
Index[4]	値



5. 配列を使って5回勝負の結果を出力する

5.1 フローを5回勝負に変更する

5.2 変数の追加(配列の変数)

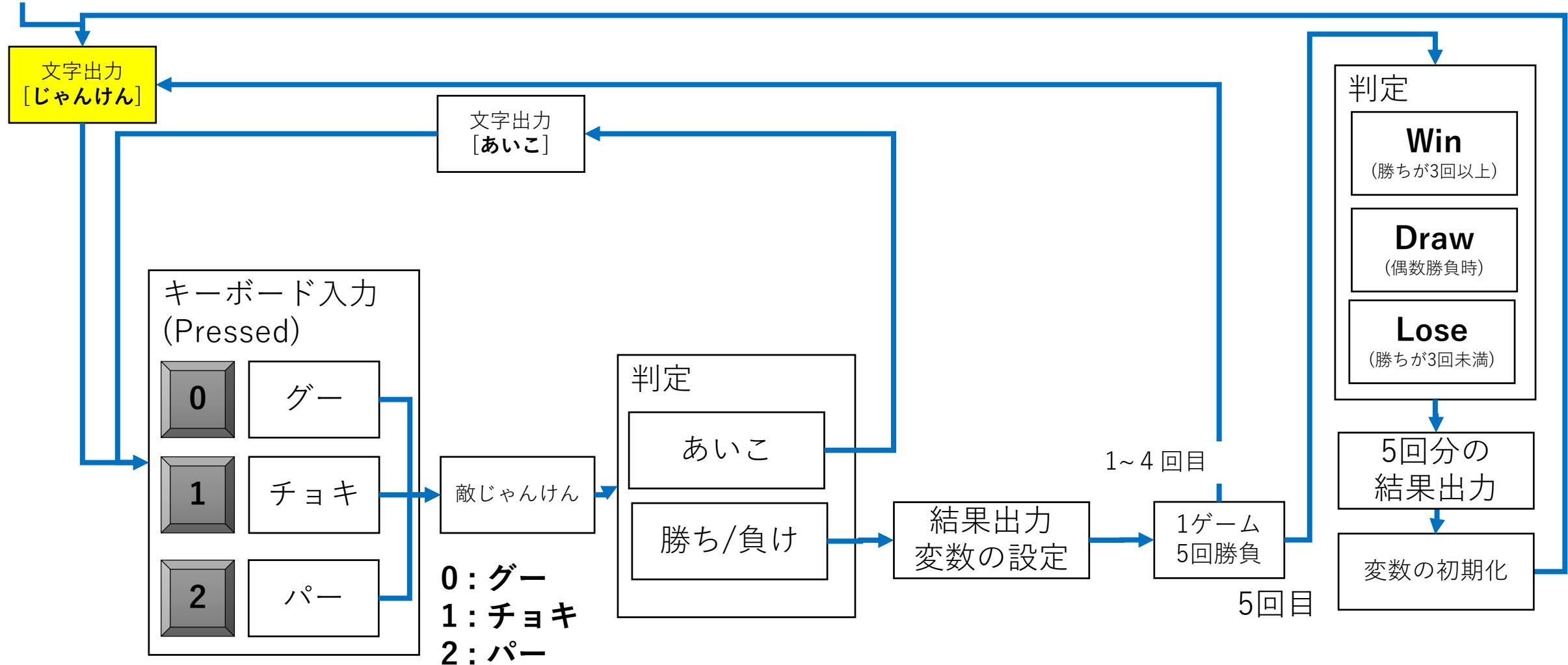
5.3 配列の操作

5.4 フローの実装

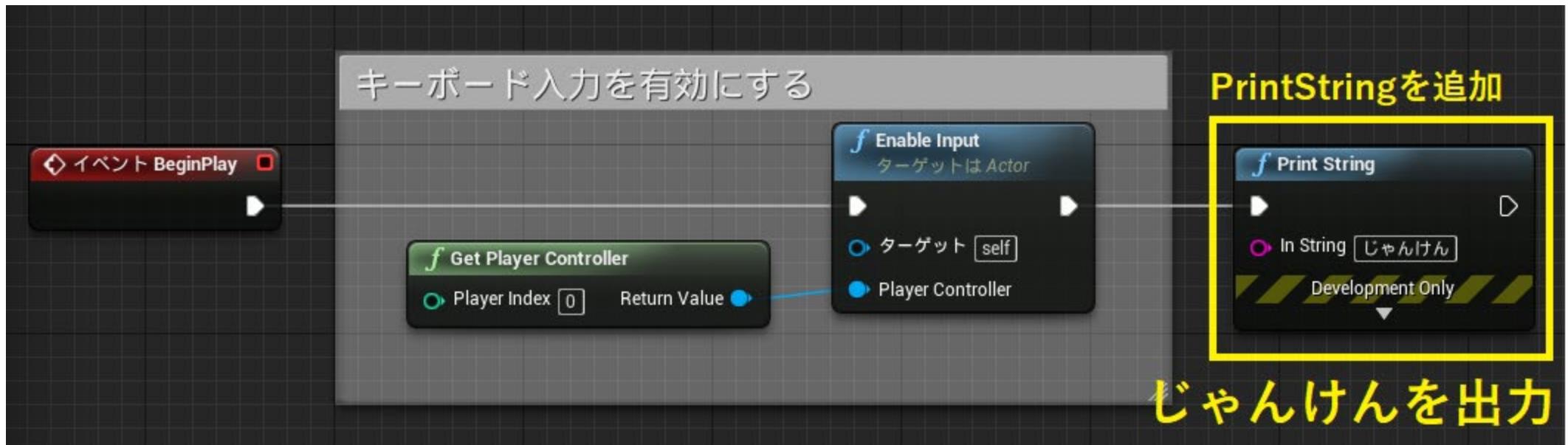
5.5 カスタムイベントを作成して処理を切り離す

開始時に[じゃんけん]を文字出力する

Start

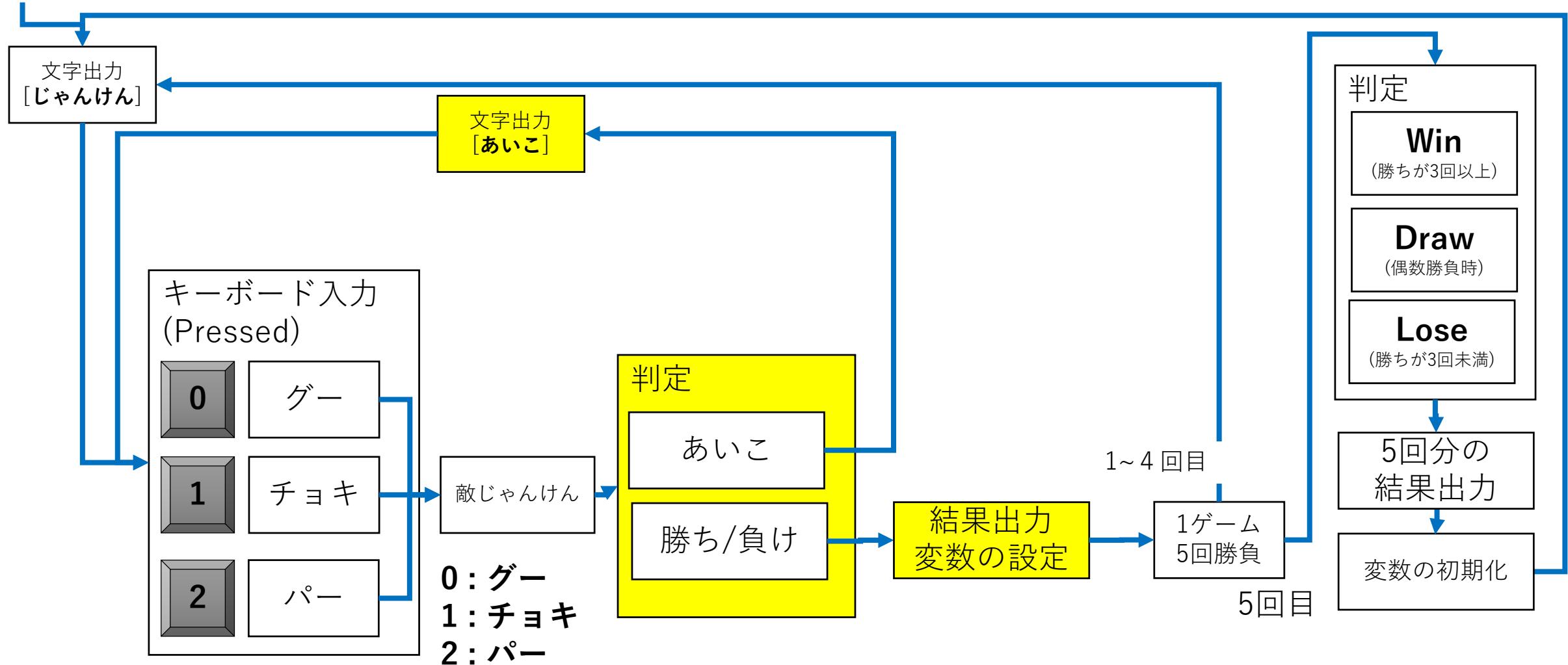


開始時に[じゃんけん]を出力するPrintStringを追加する



判定の処理を実装

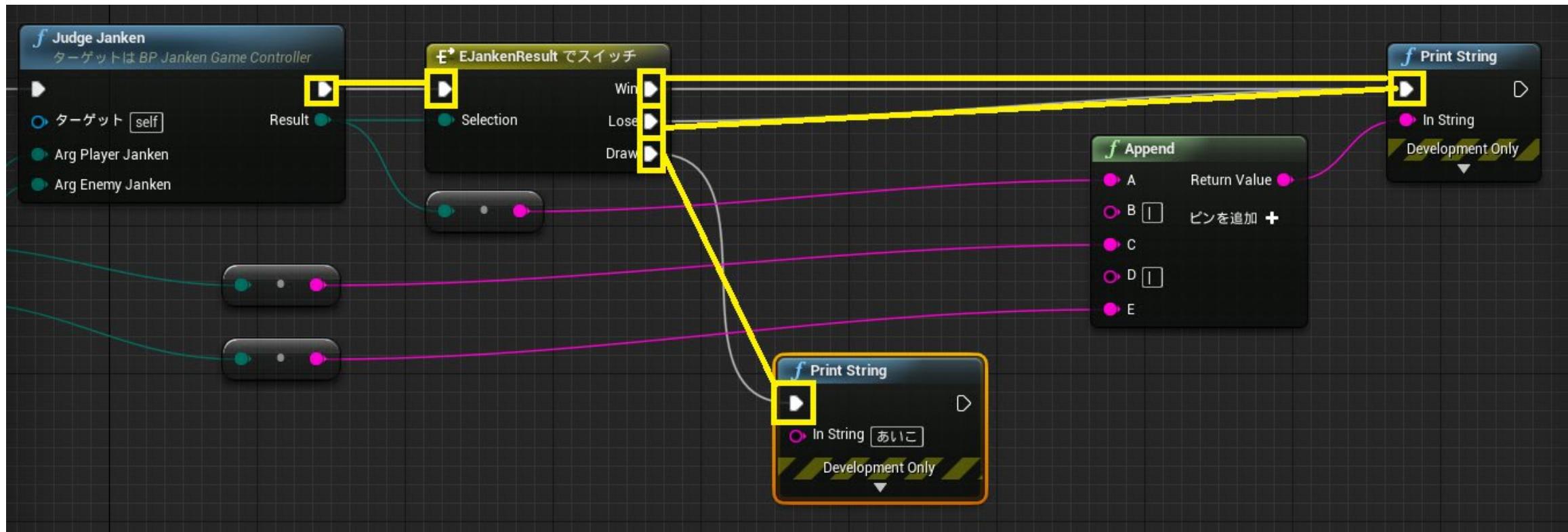
Start



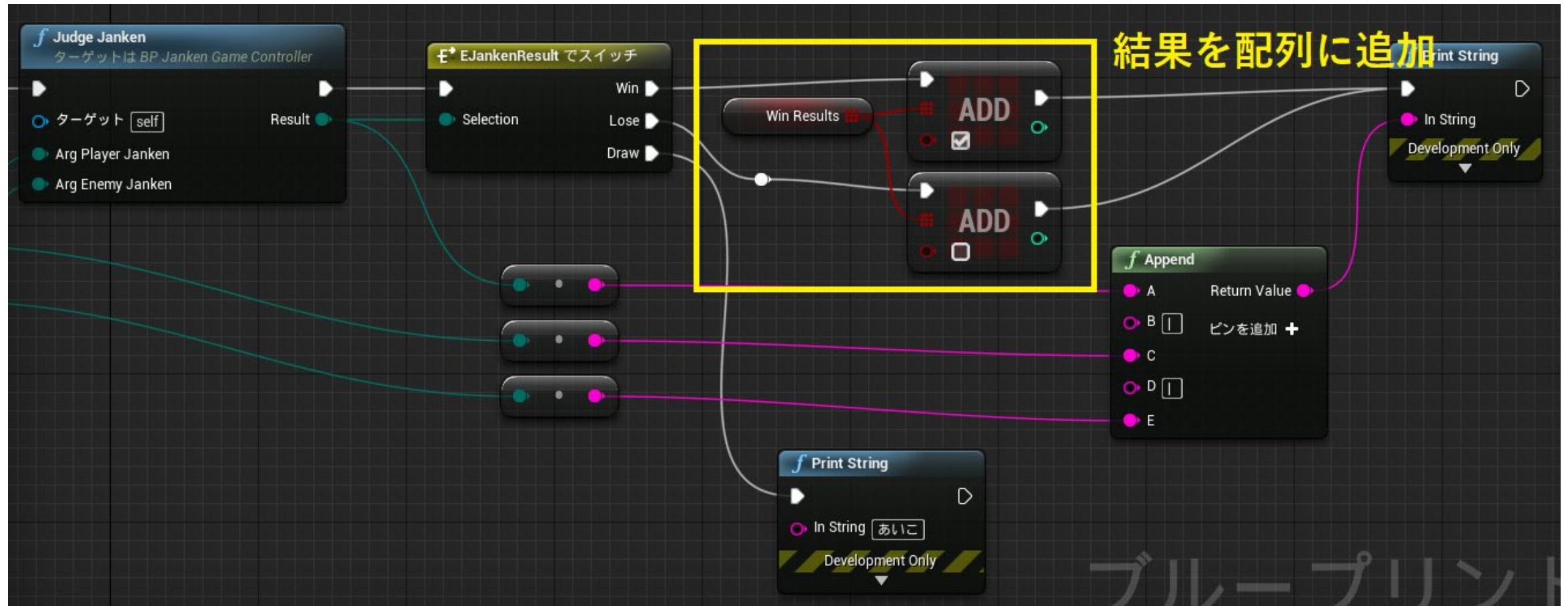
JudgeJankenResultのスイッチを追加



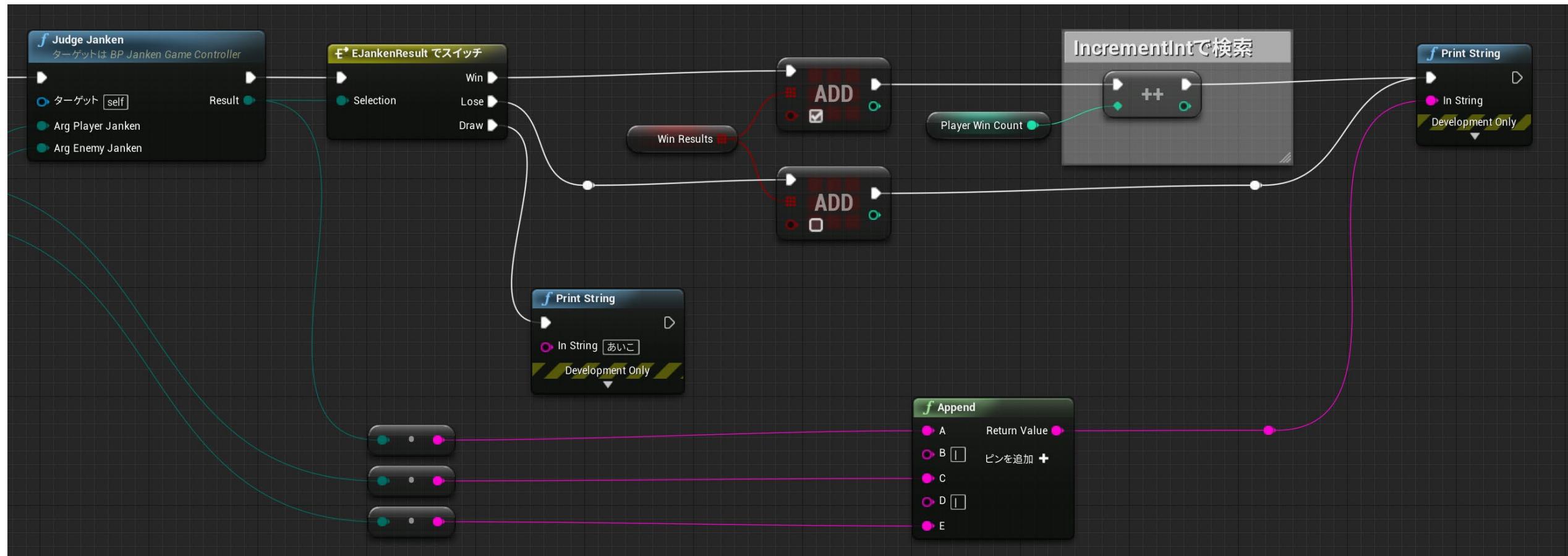
スイッチの分岐処理を実装



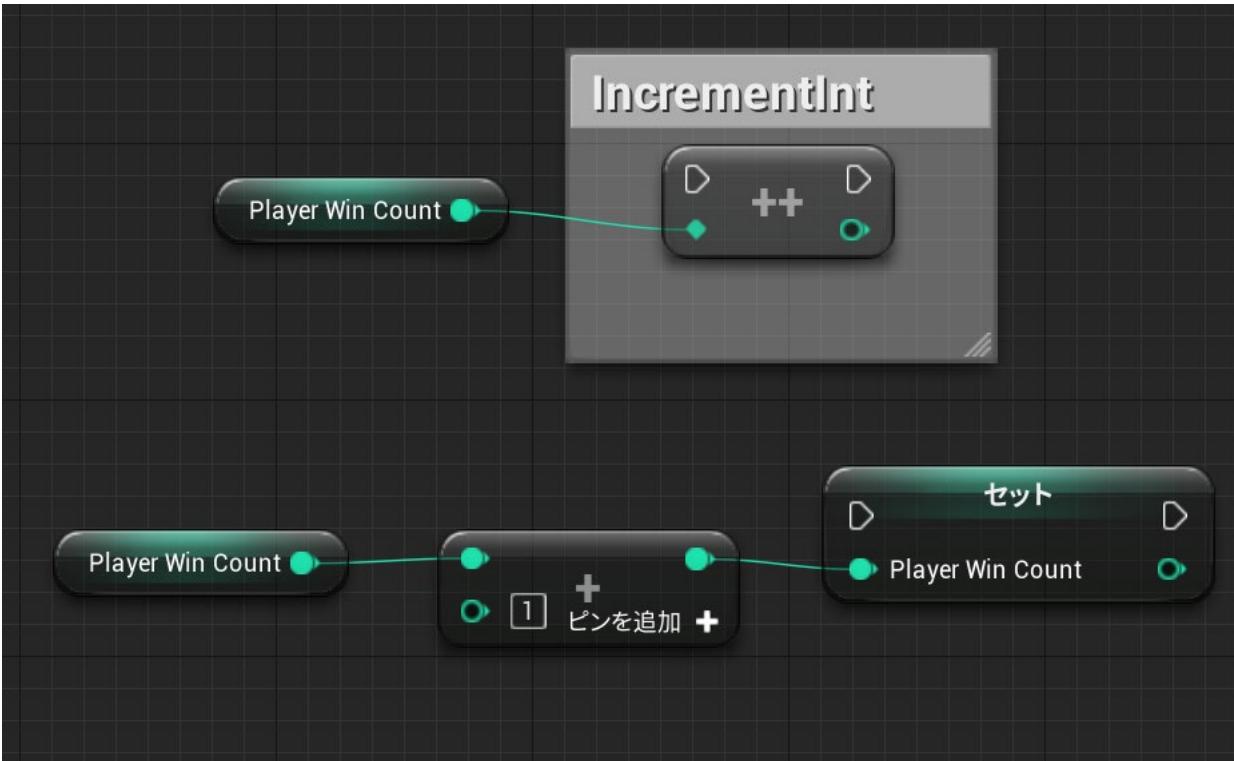
Win,Loseの場合は結果を配列bWinResultsに追加



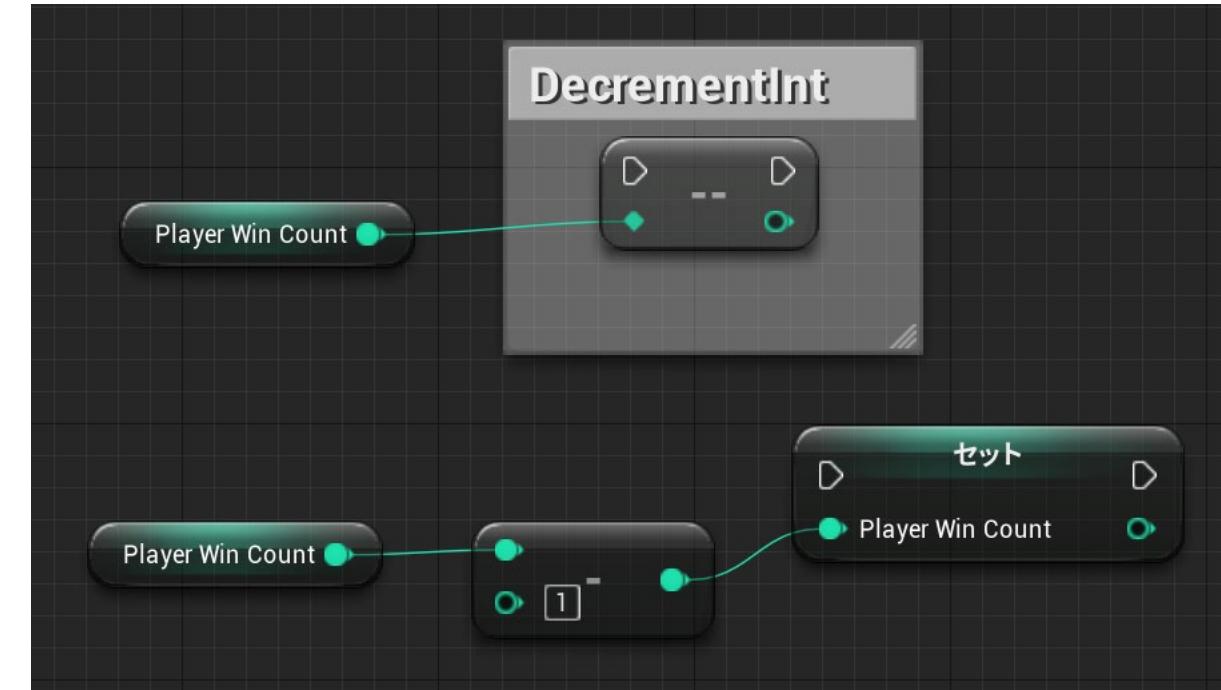
Winの場合はPlayerWinCountを+1する



IncrementIntを使うと+1するのが楽
DecrementIntは-1をする

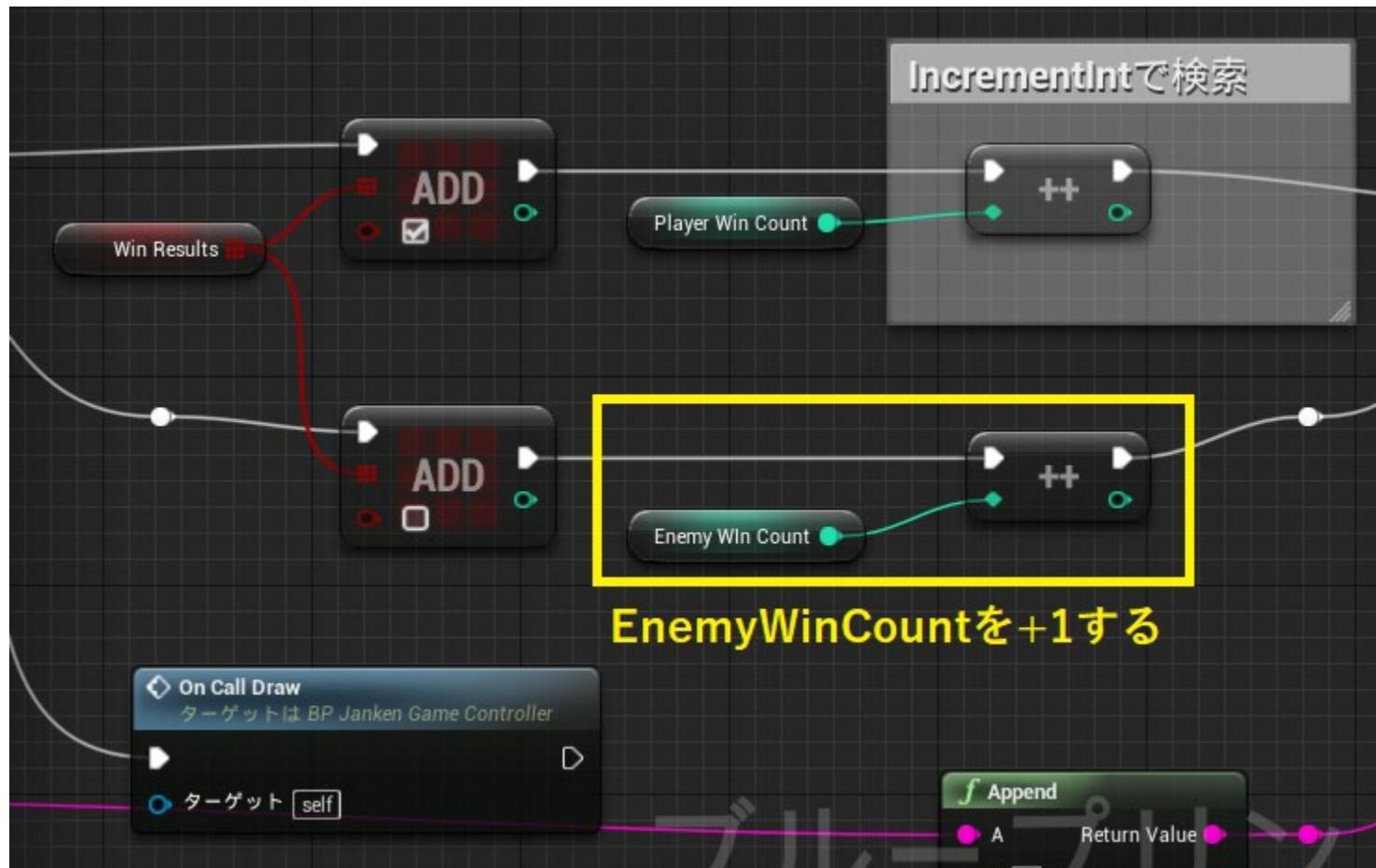


IncrementIntを使えば、
Setノードを使わずに+1することが出来る



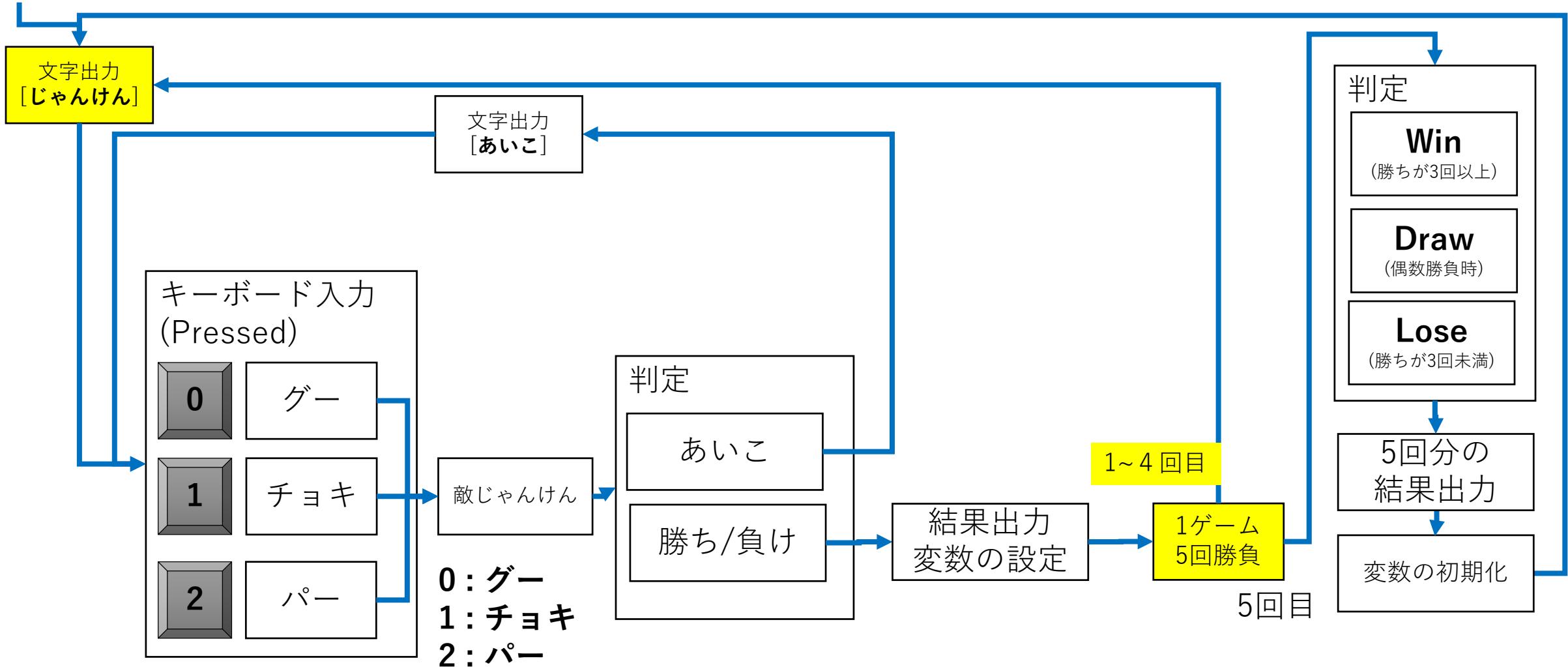
DecrementIntを使えば、
Setノードを使わずに-1することが出来る

Loseの場合はEnemyWinCountを+1する

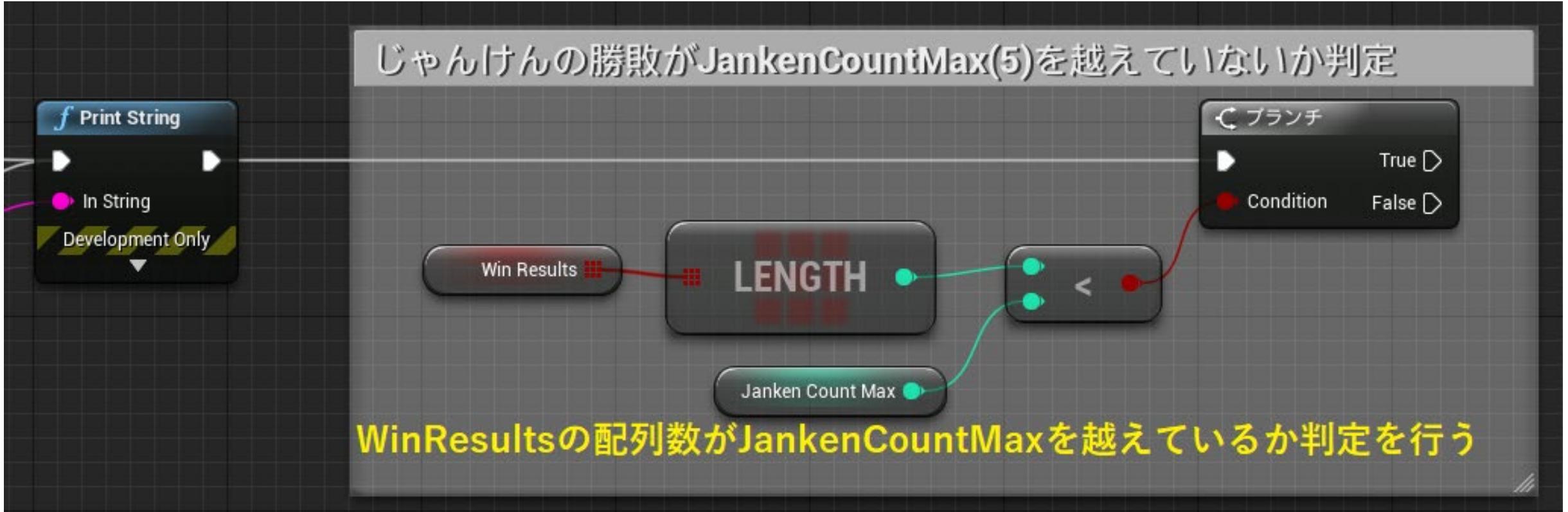


判定の処理を実装

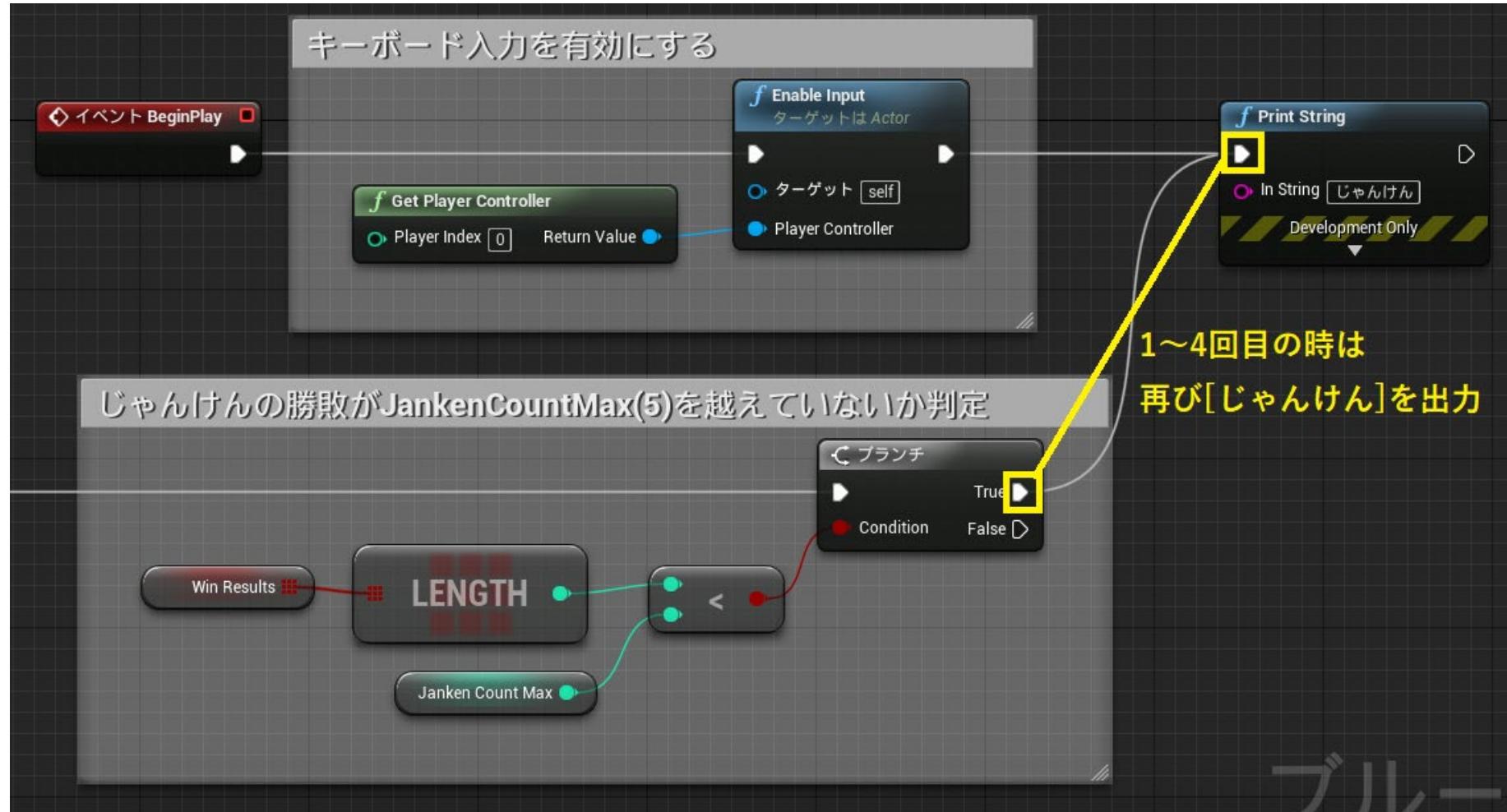
Start



じゃんけんの勝敗がJankenCountMax(5)を越えていないか確認

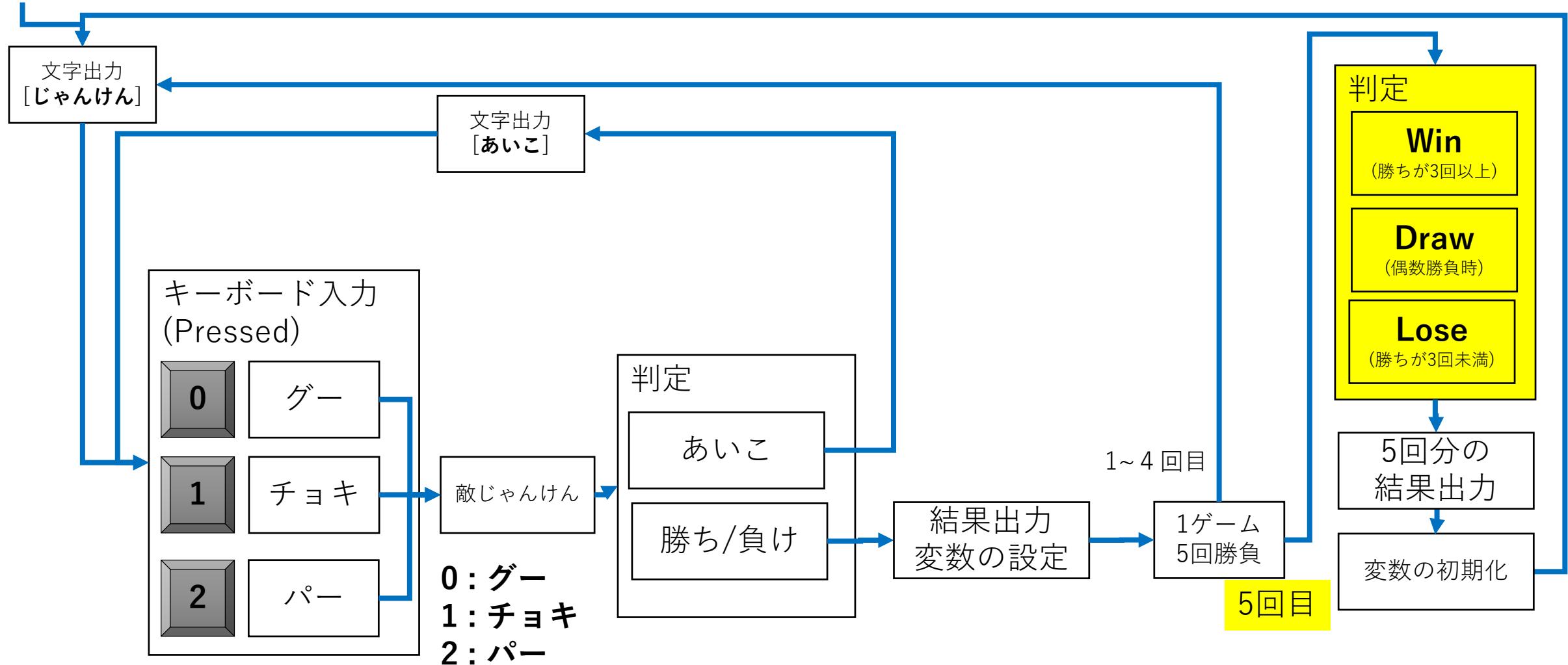


1~4回目の時は再び[じゃんけん]を出力

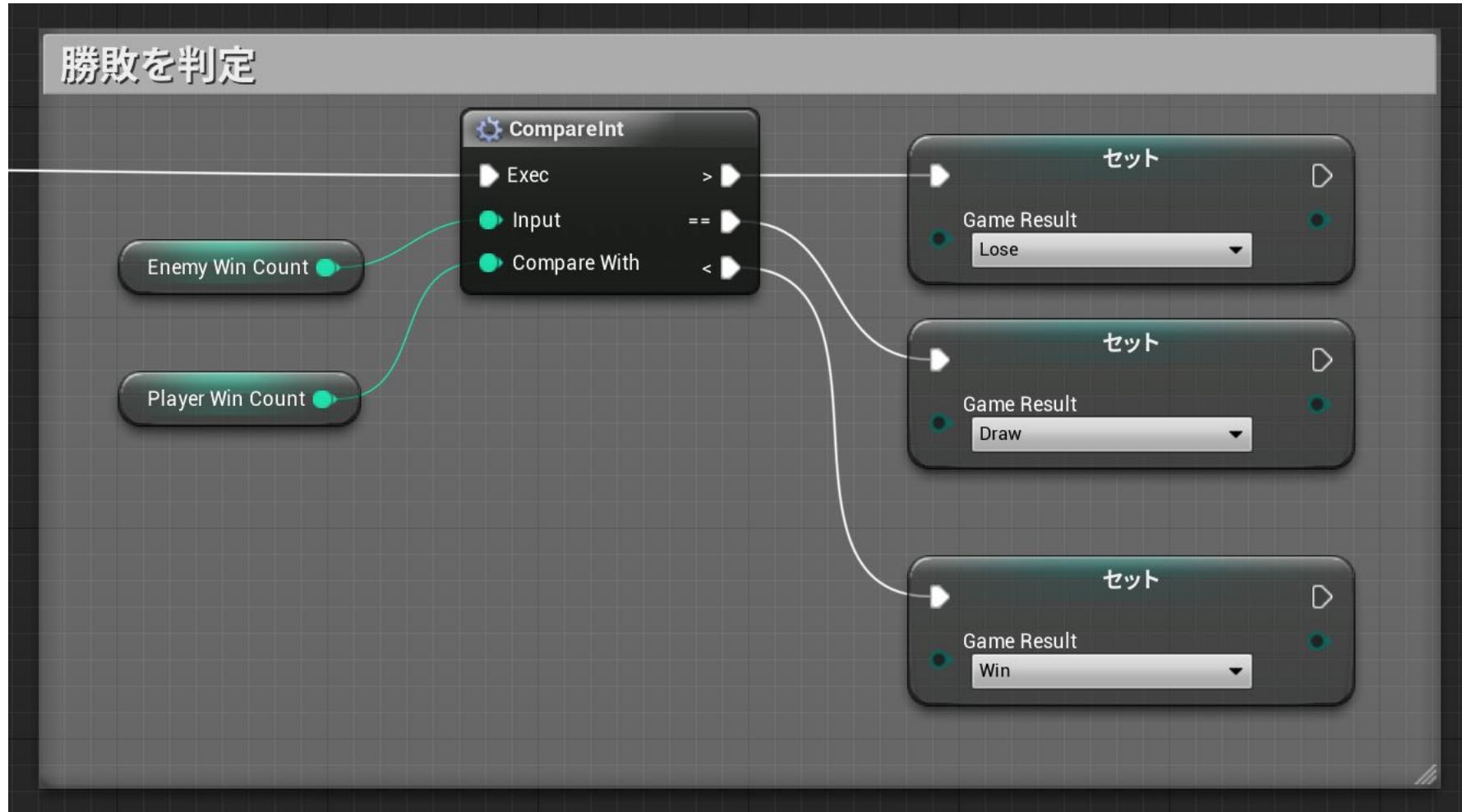


5回目の勝敗が完了した時の判定の処理を実装

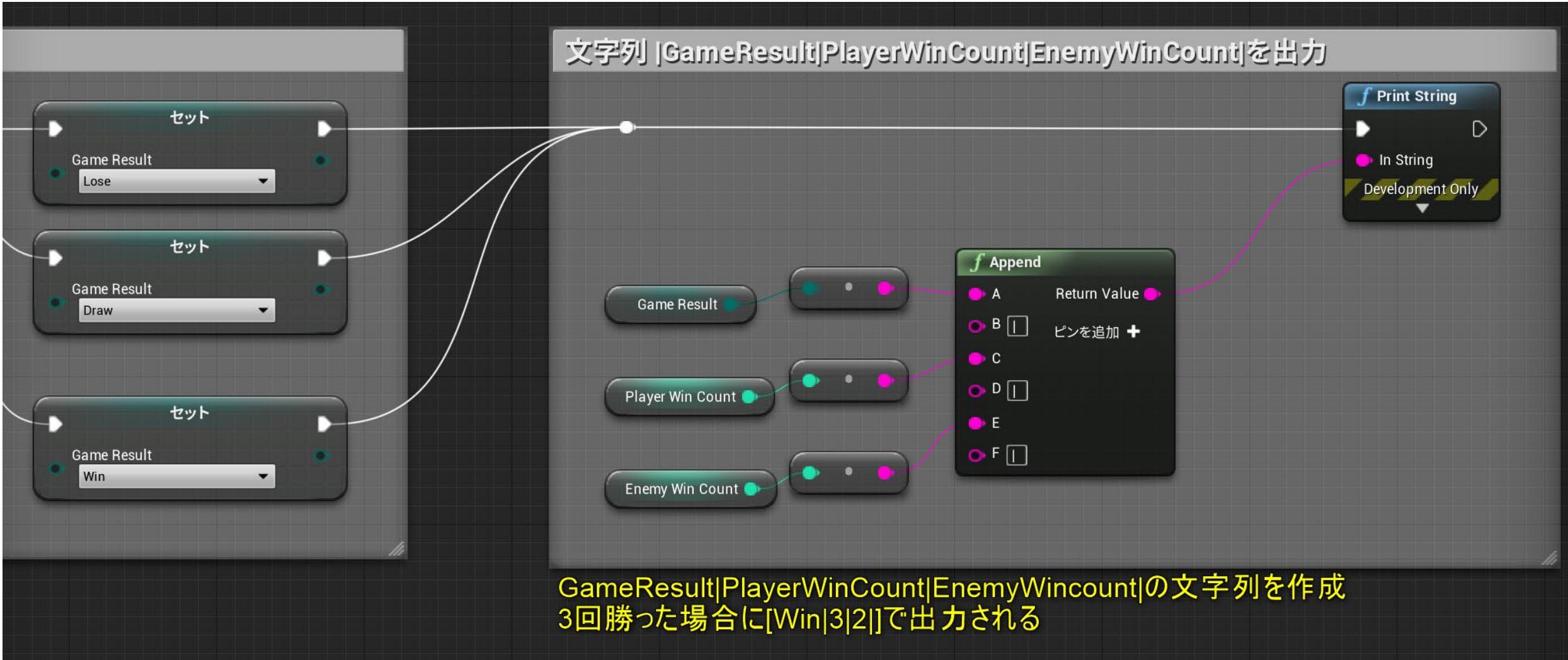
Start



勝敗を判定する

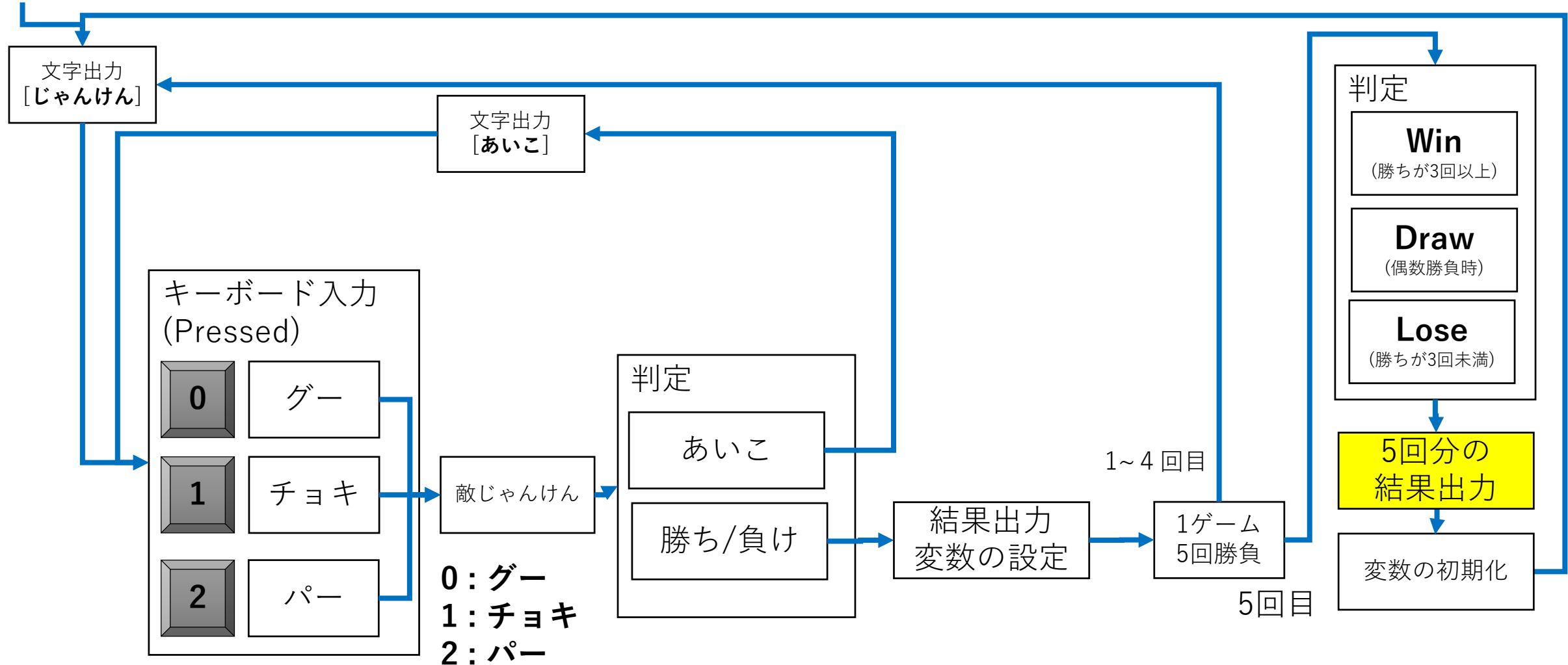


勝敗が分かる文字列を作成して出力する



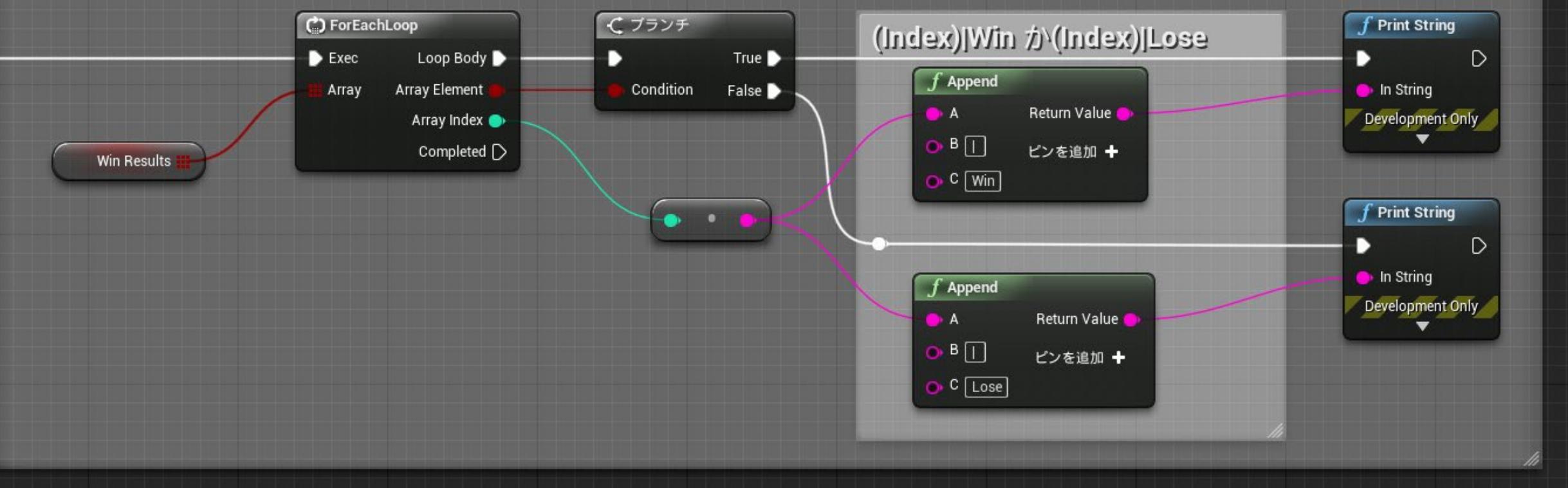
5回分の結果出力

Start



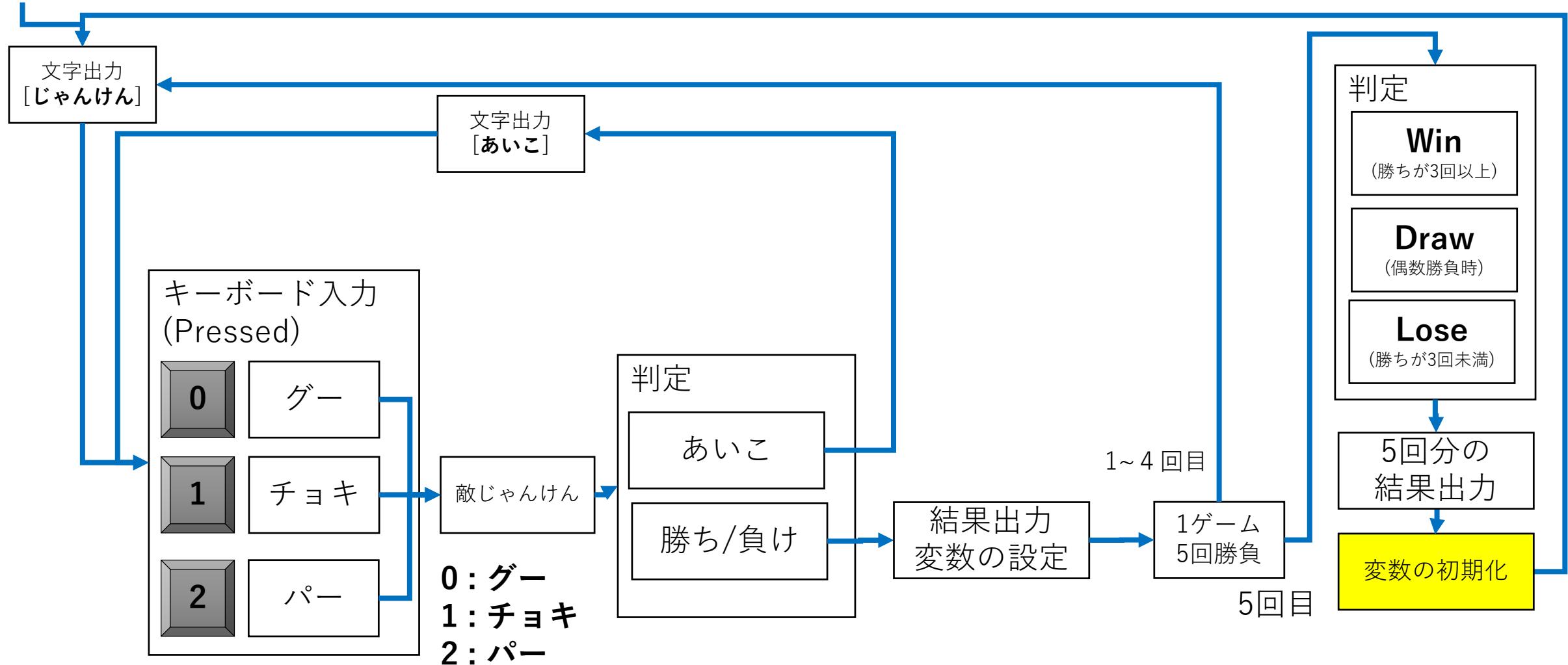
5回分の勝敗を出力する

5回分の勝敗を出力する

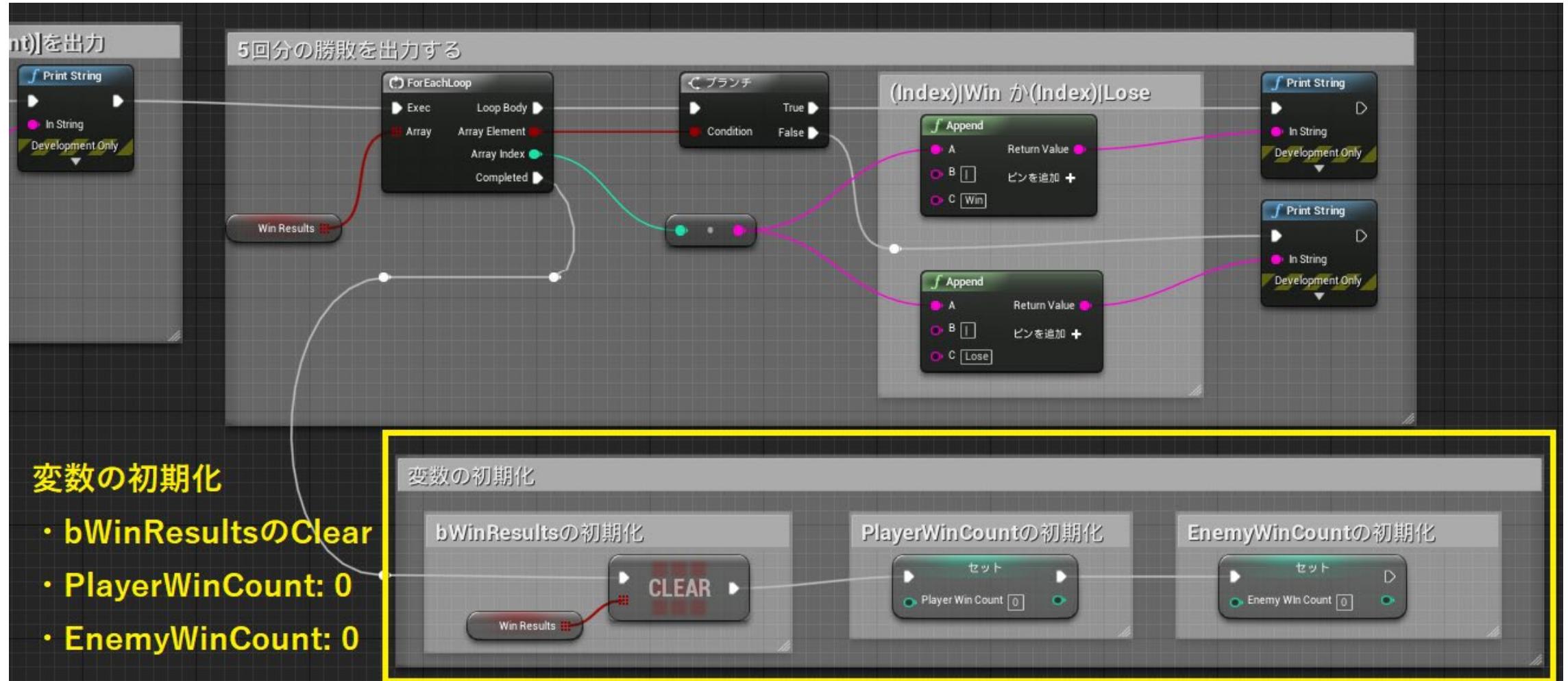


変数の初期化を行う

Start

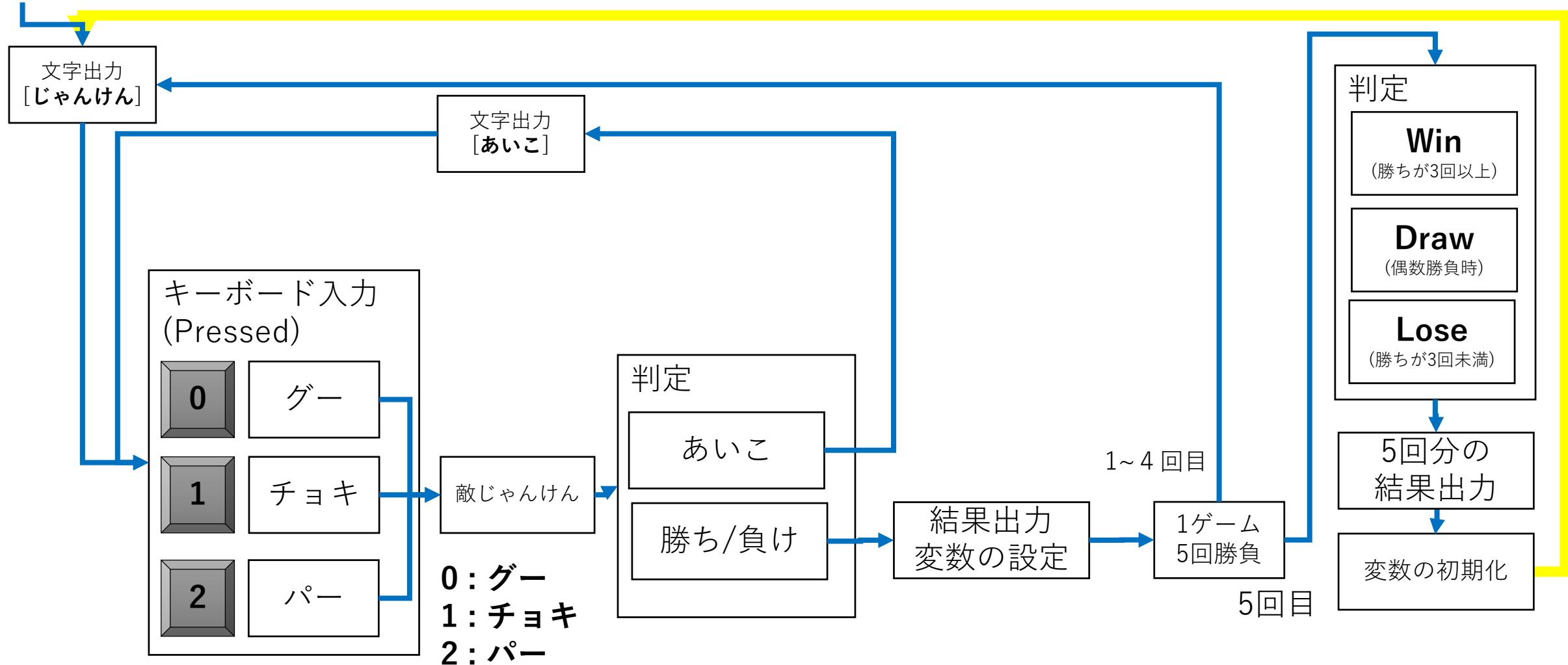


変数の初期化

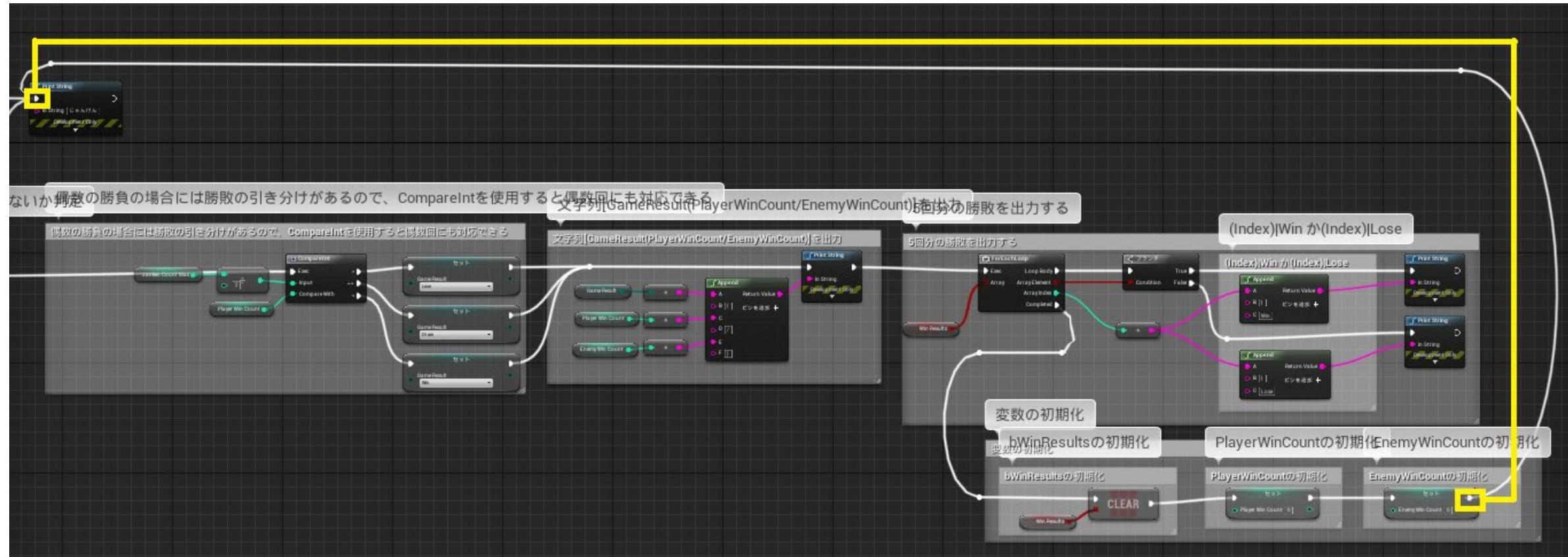


[じゃんけん]を出力する

Start



初期化処理の終わりと[じゃんけん]を出力するPrintStringとつなげる



プレイして確認する



5. 配列を使って5回勝負の結果を出力する

5.1 フローを5回勝負に変更する

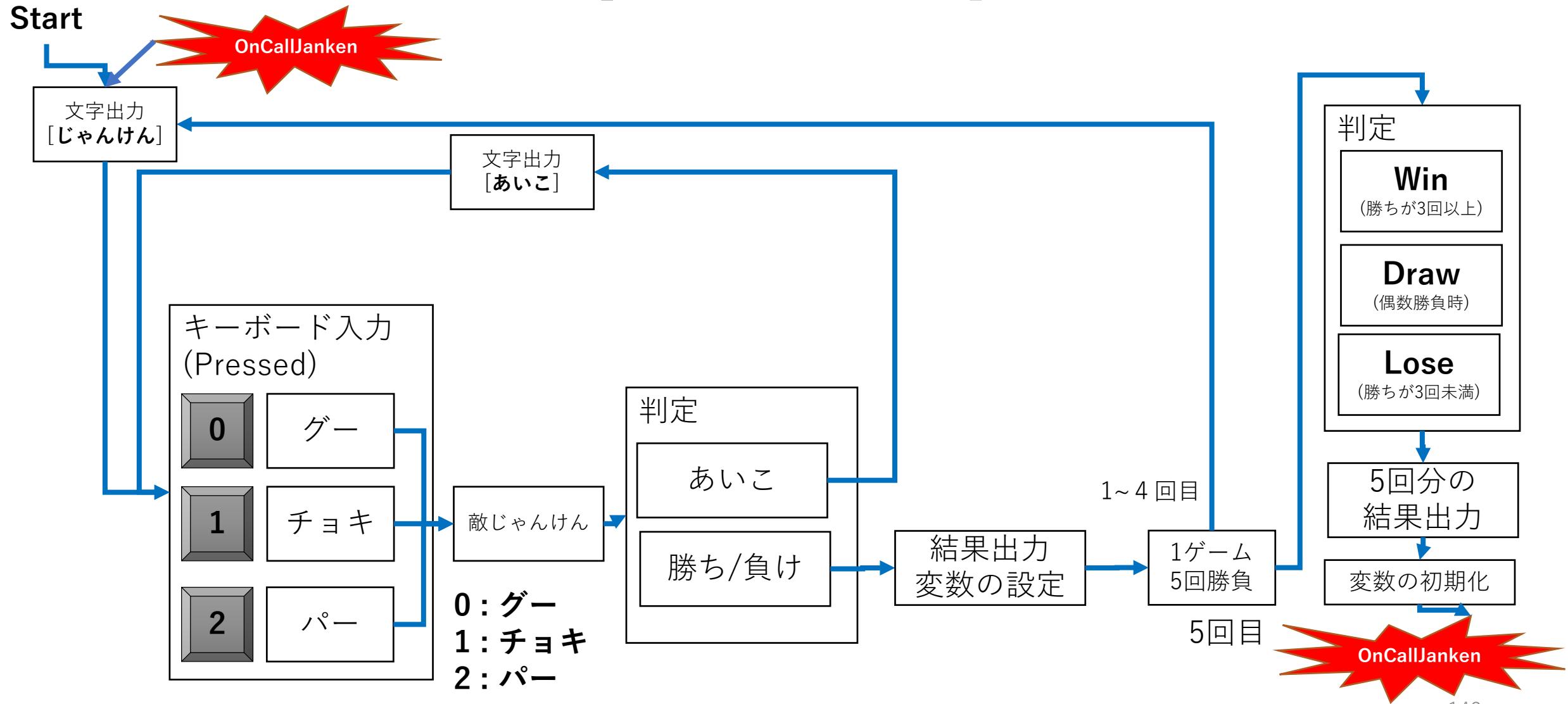
5.2 変数の追加(配列の変数)

5.3 配列の操作

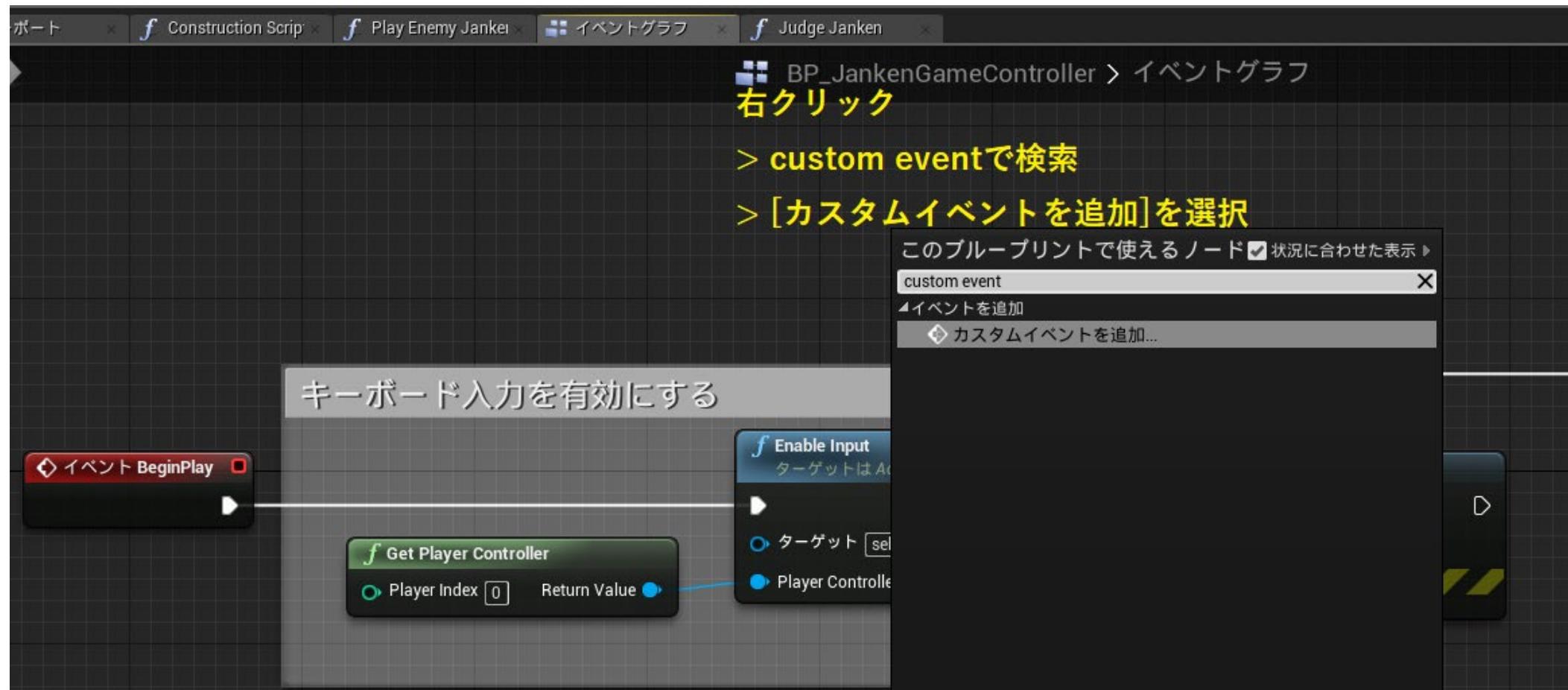
5.4 フローの実装

5.5 カスタムイベントを作成して処理を切り離す

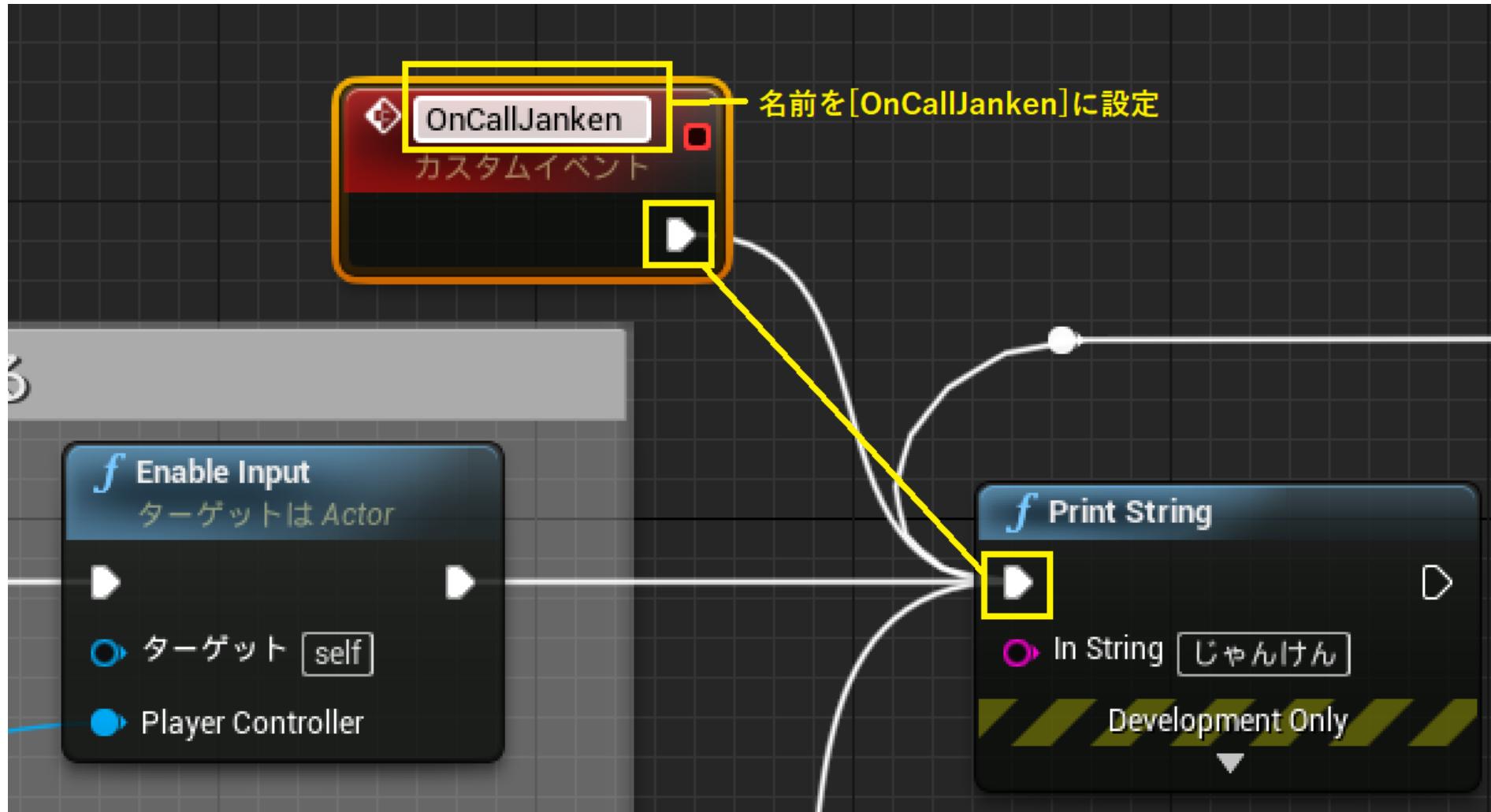
カスタムイベント[OnCallJanken]を作成



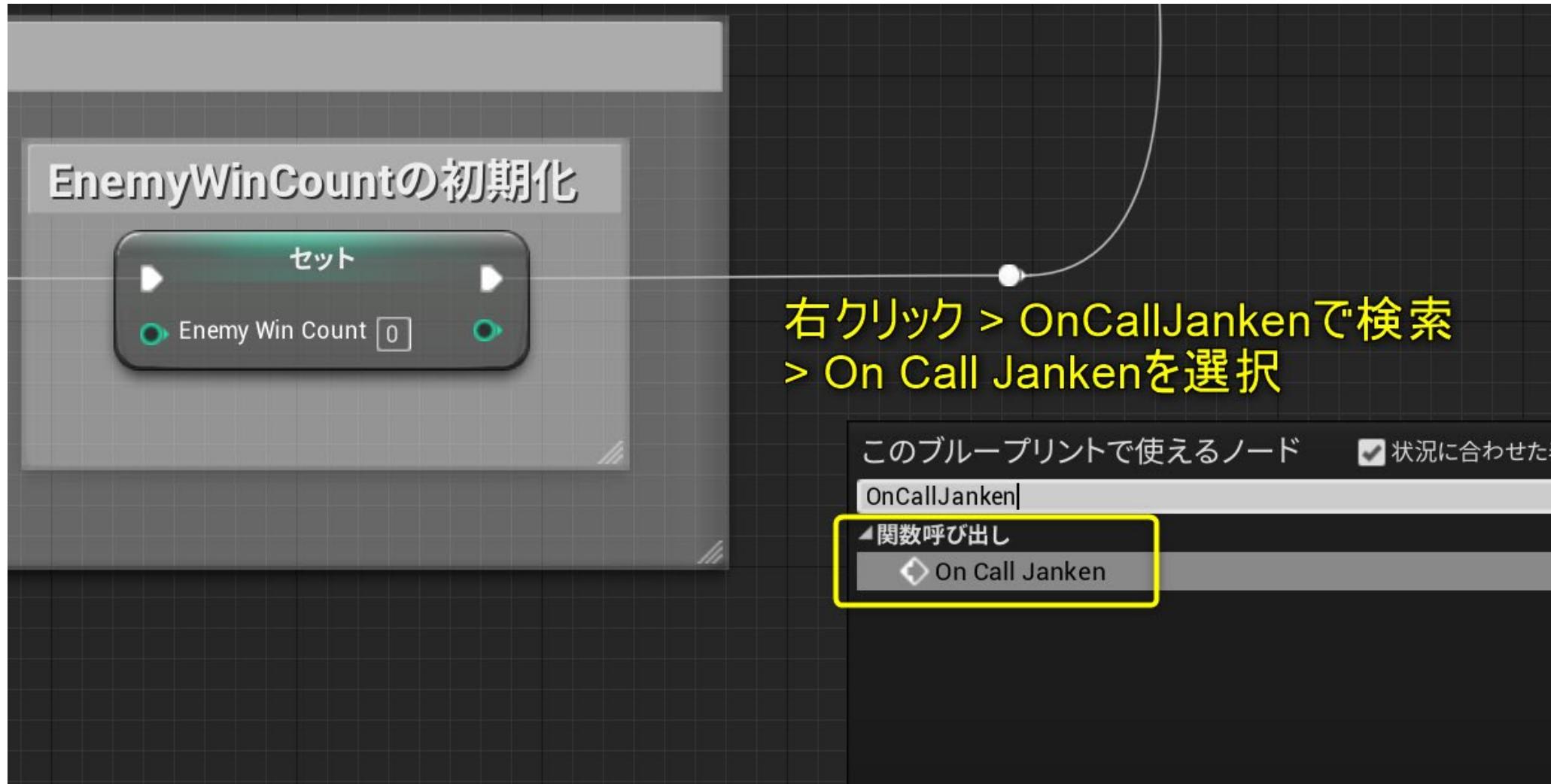
カスタムイベントを追加



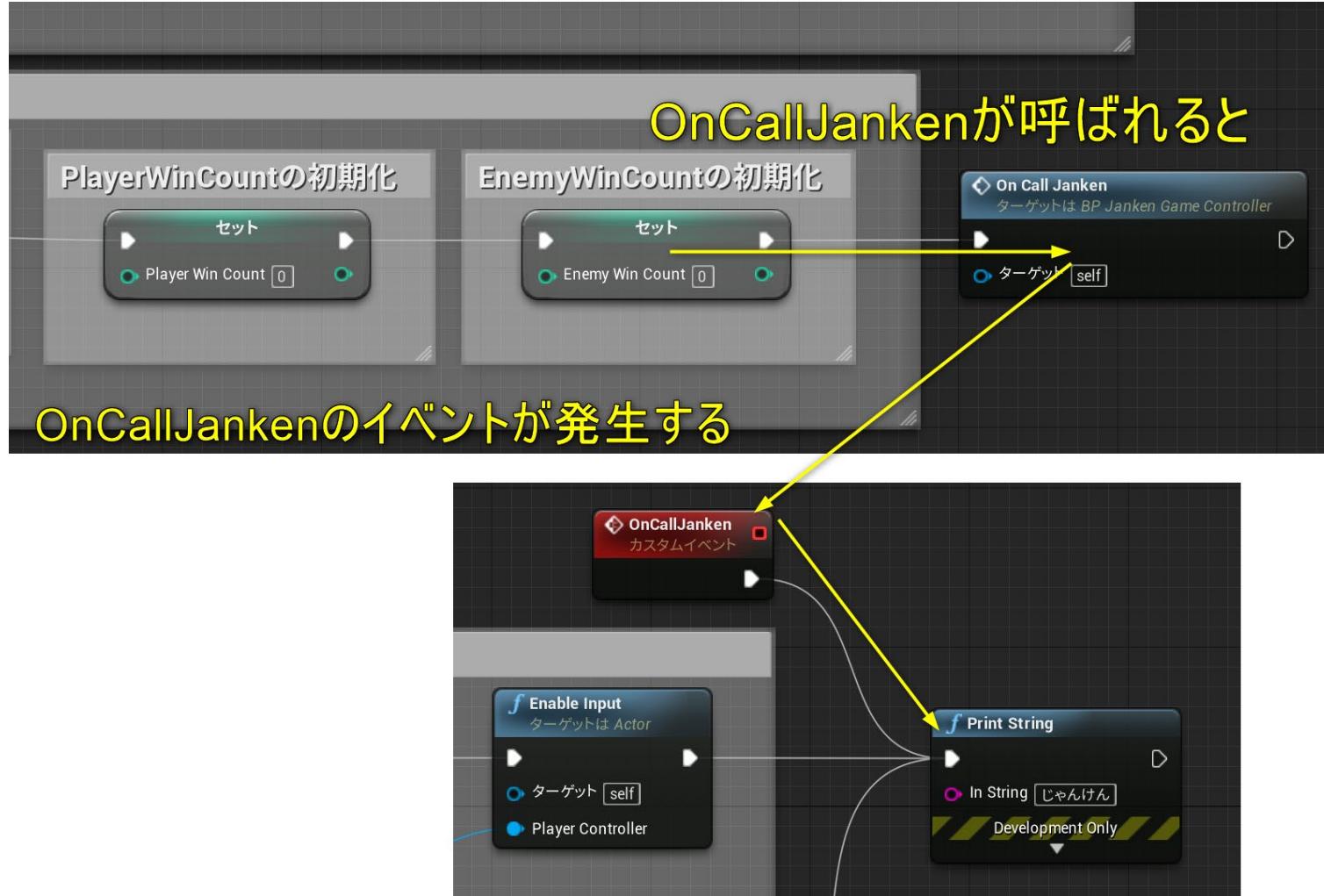
名前をOnCallJankenに設定
実行ピンをPrint Stringに接続



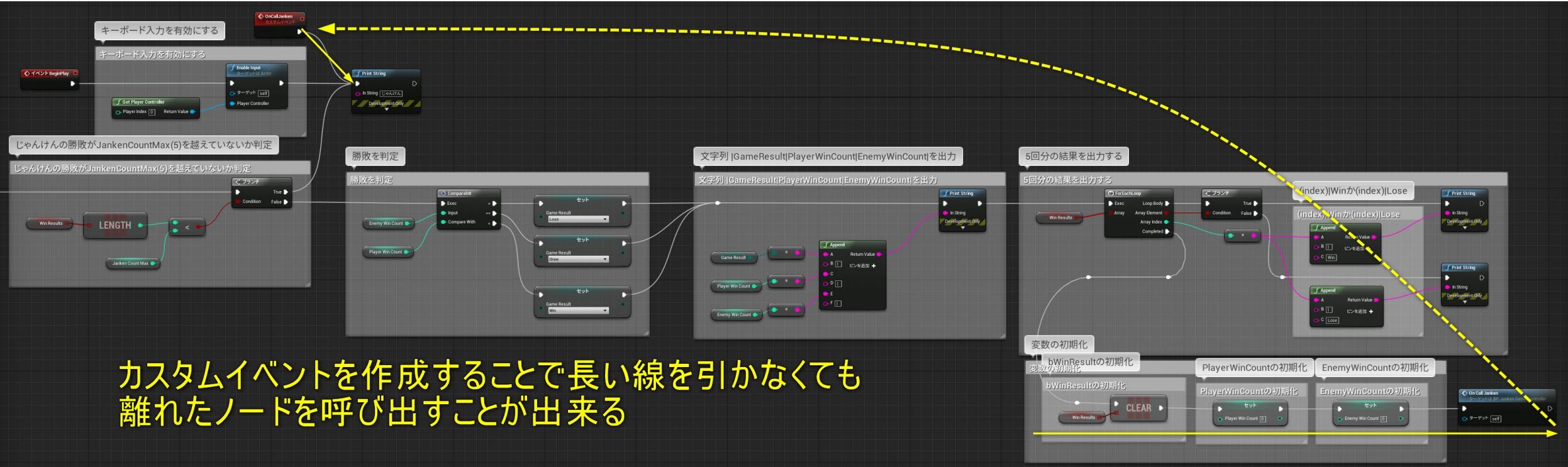
OnCallJankenを追加する



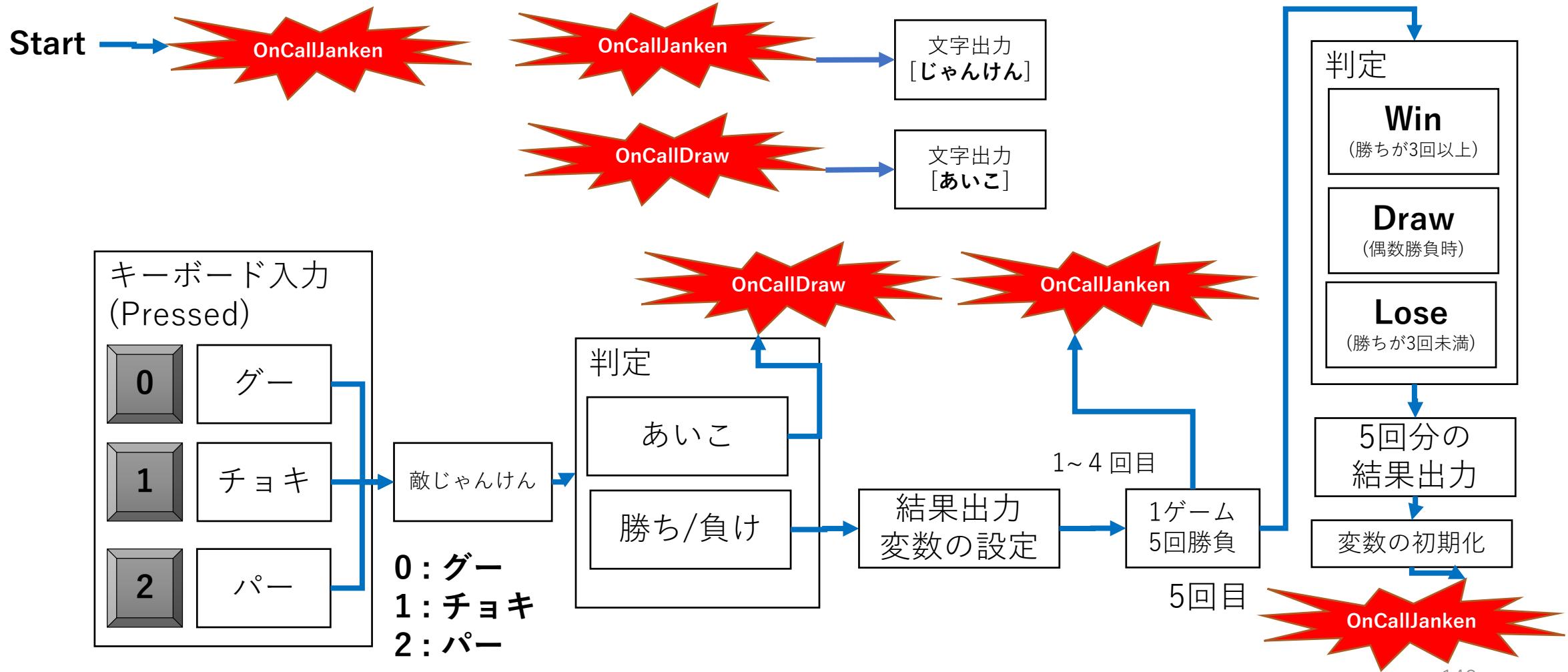
OnCallJankenが呼ばれた時に、
OnCallJankenイベントが発生する



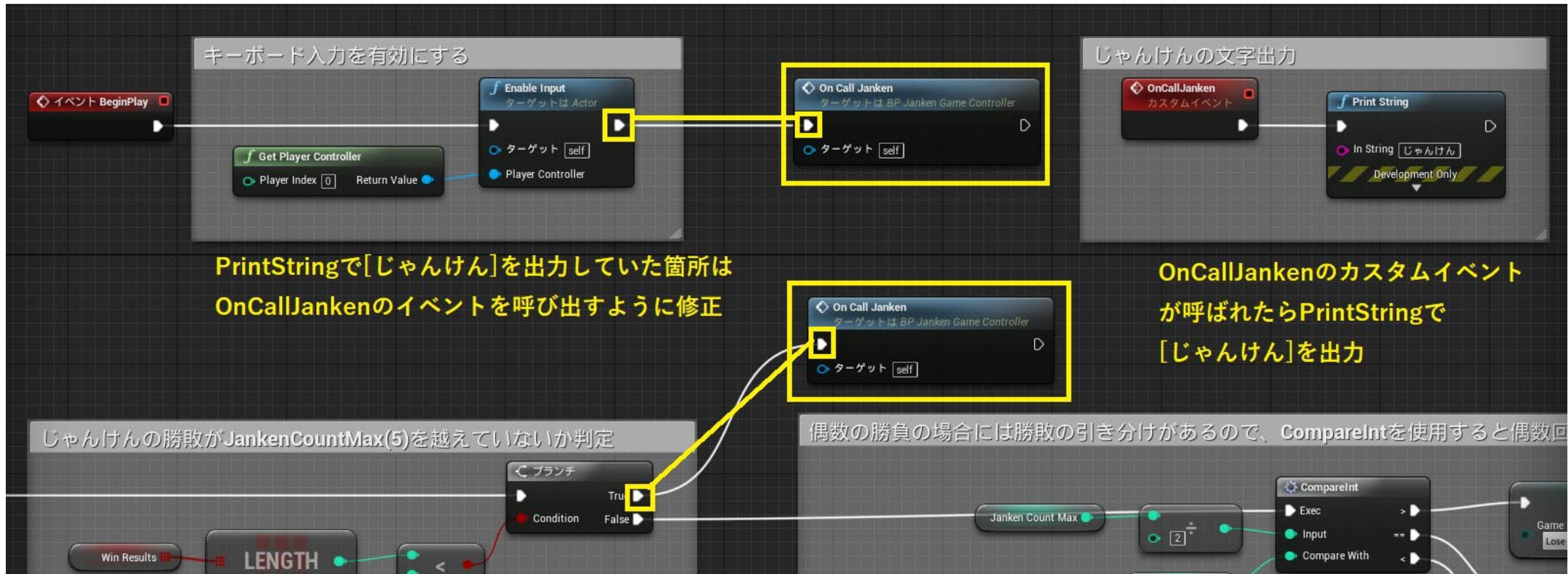
カスタムイベントを作ることで、
長い線を引かなくても、離れたノードを呼び出すことが出来る



OnCallDrawも追加
カスタムイベントで処理を切り離す



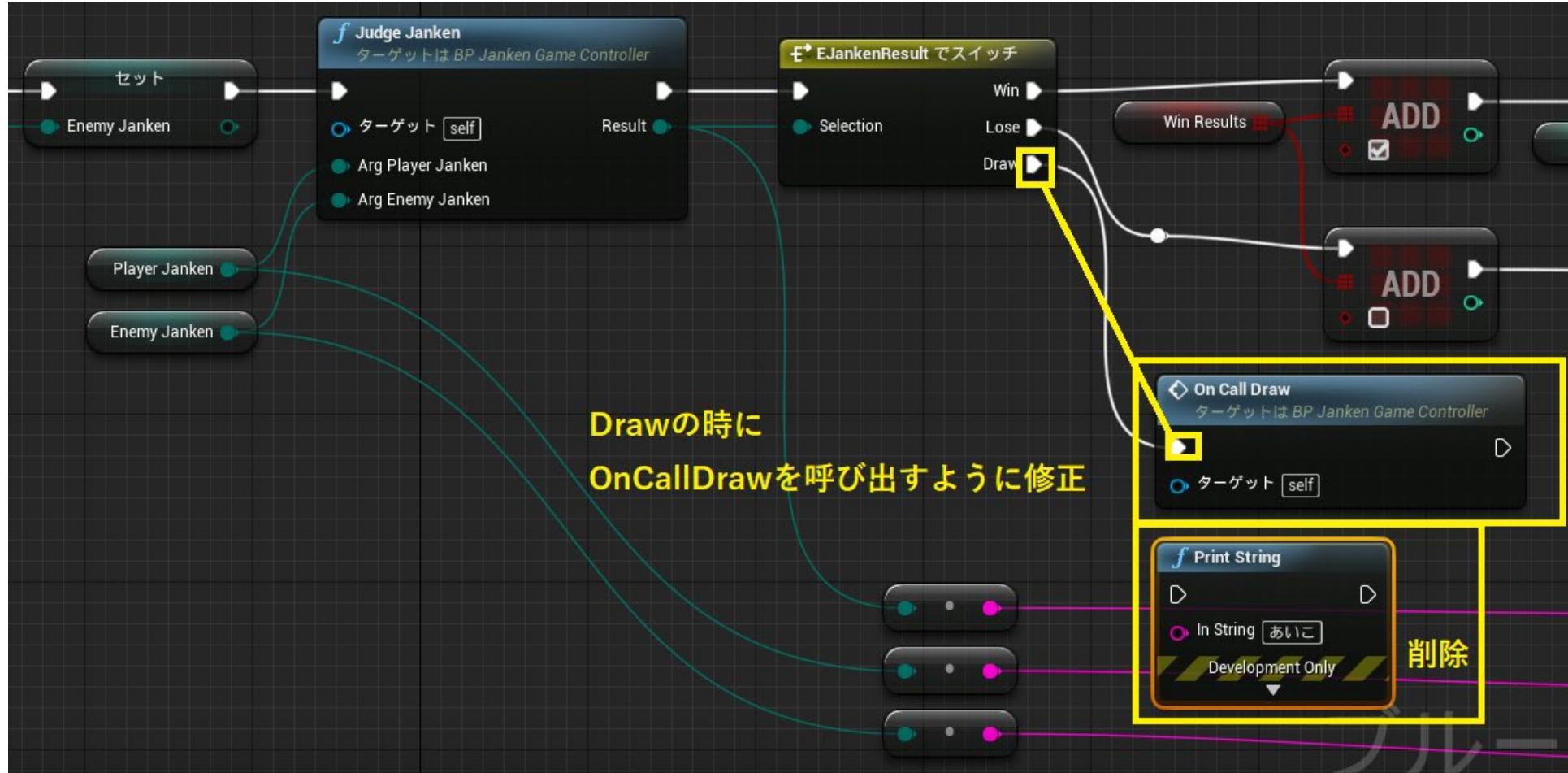
OnCallJankenのカスタムイベントが呼ばれたら[じゃんけん]を出力
PrintStringで[じゃんけん]を出力していた箇所はOnCallJankenのイベントを呼び出すように修正する



カスタムイベント:OnCallDrawを作成



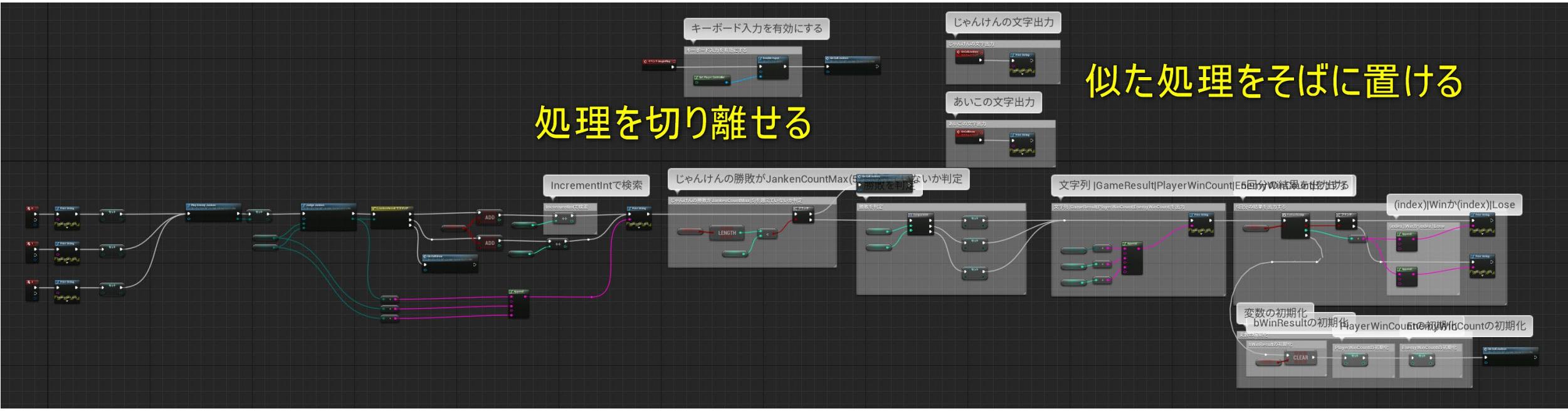
じゃんけんの結果がDrawの時に OnCallDrawを呼び出すように修正



カスタムイベントで処理を切り離すことが出来るようになり、似た処理を近くに置くことが出来る

処理を切り離せる

似た処理をそばに置ける



切り離したことによって、関連が分かりづらくなるので、カスタムイベントについて知らないと処理を読むことが出来ない

プレイして確認する



6. 構造体を使って出力する結果の情報を増やす

6. 構造体を使って出力する結果の情報を増やす

6.1 構造体:FJankenResultを作成する

6.2 変数の追加(構造体の配列の変数)

6.3 bWinResultsを使用している箇所をFJankenResultsに置き換える

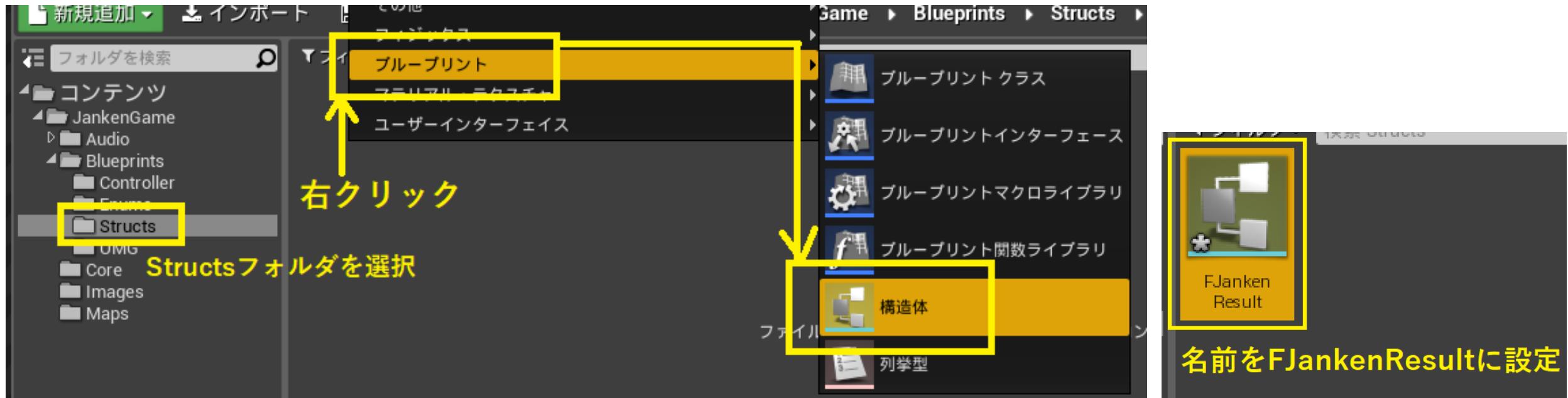
6. 構造体を使って出力する結果の情報を増やす

6.1 構造体:FJankenResultを作成する

6.2 変数の追加(構造体の配列の変数)

6.3 bWinResultsを使用している箇所をFJankenResultsに置き換える

構造体:FJankenResultを作成する



Structs フォルダを選択

> 右クリック

> ブループリント

> 構造体

名前をFJankenResultに設定

構造体の変数を設定する



FJankenResultをダブルクリックで開く
> 変数が3つになるように[新規変数]をクリック
> 変数名、変数の型を設定

変数名	変数の型	デフォルト値
bWin	Boolean	-(設定なし)
PlayerJanken	EJanken	-(設定なし)
EnemyJanken	EJanken	-(設定なし)

6. 構造体を使って出力する結果の情報を増やす

6.1 構造体:FJankenResultを作成する

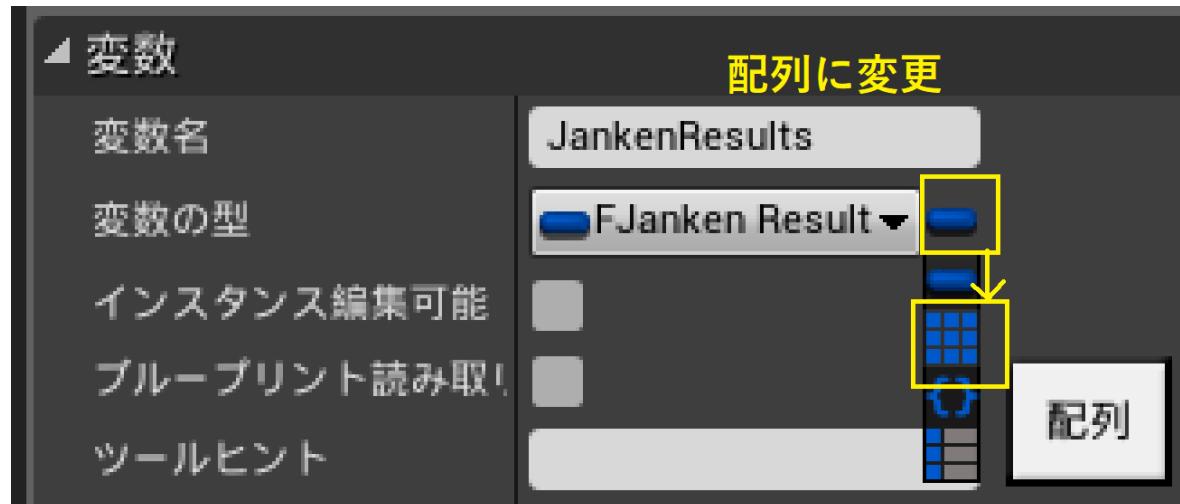
6.2 変数の追加(構造体の配列の変数)

6.3 bWinResultsを使用している箇所をFJankenResultsに置き換える

変数JankenResultsを追加する 変数の型はFJankenResultに設定



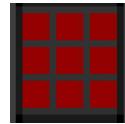
変数の型を配列に変更する



变数名	变数の型	デフォルト 値
PlayerJanken	EJanken	-(設定なし)
EnemyJanken	EJanken	-(設定なし)
bWinResults (削除予定)	Boolean (配列)	-(設定なし)
WinCount	Integer	-(設定なし)
JankenCountMax	Integer	5
GameResult	EJankenResult	-(設定なし)
JankenResults	FJankenResult (配列)	-(設定なし)

bWinResultsを拡張したのがJankenResultsなので、処理を構造体に移行出来たらbWinResultsを削除する予定

Booleanの配列と構造体の配列の違い



Booleanの配列

Index[0]	値
Index[1]	値
Index[2]	値
Index[3]	値
Index[4]	値

1つのIndexに同じ型の値しか持てない



Index[0]	bWin	値
	PlayerJanken	値
	EnemyJanken	値
Index[1]	bWin	値
	PlayerJanken	値
	EnemyJanken	値
Index[2]	bWin	値
	PlayerJanken	値
	EnemyJanken	値
Index[3]	bWin	値
	PlayerJanken	値
	EnemyJanken	値
Index[4]	bWin	値
	PlayerJanken	値
	EnemyJanken	値

1つのIndexに異なる型の値を複数持つことが出来る

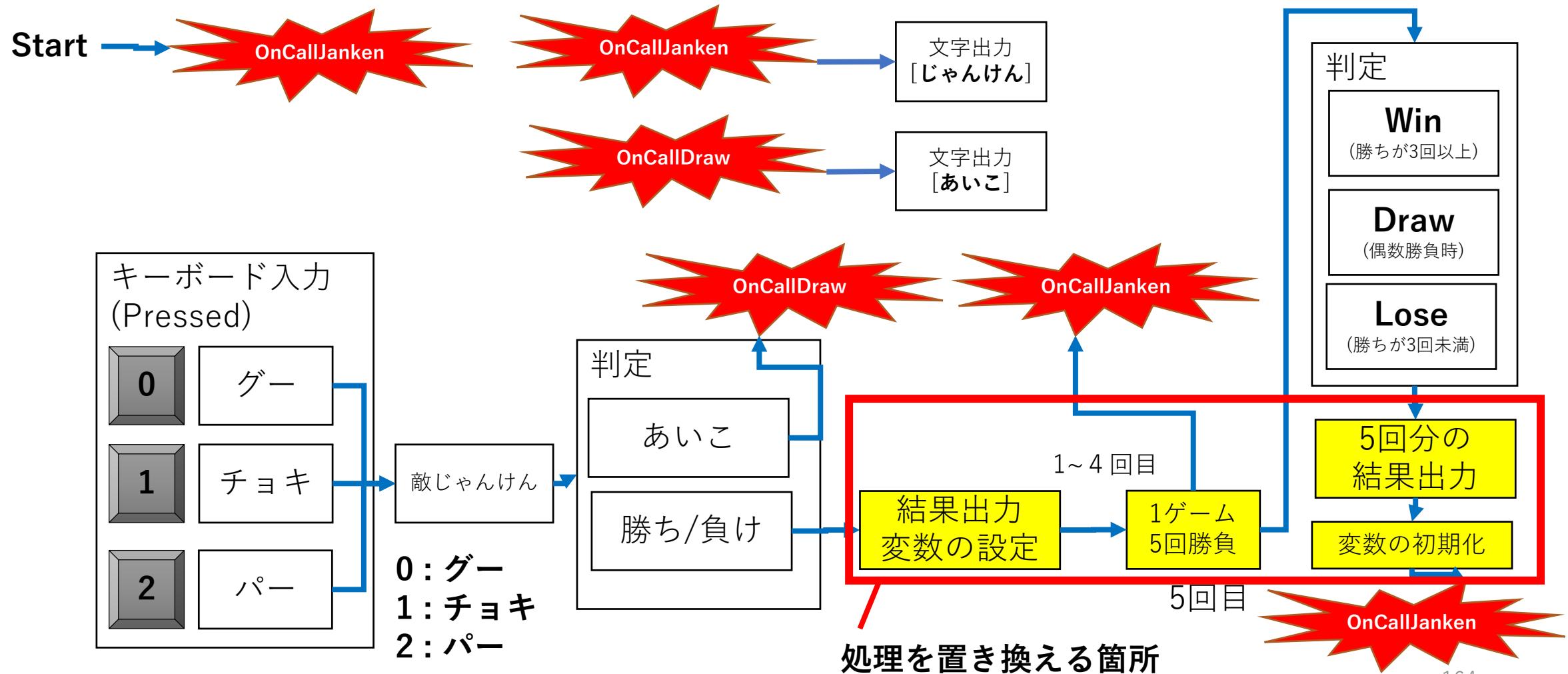
6. 構造体を使って出力する結果の情報を増やす

6.1 構造体:FJankenResultを作成する

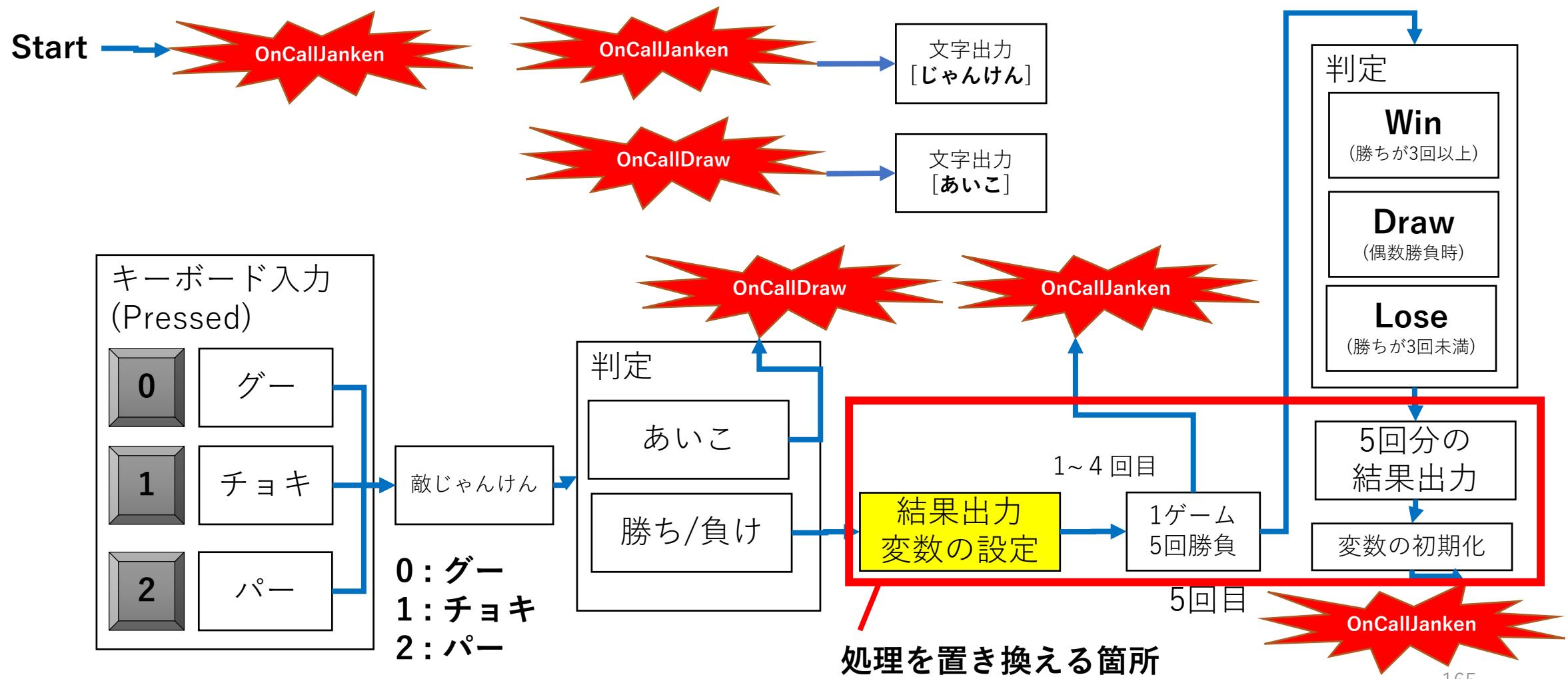
6.2 変数の追加(構造体の配列の変数)

6.3 bWinResultsを使用している箇所をFJankenResultsに置き換える

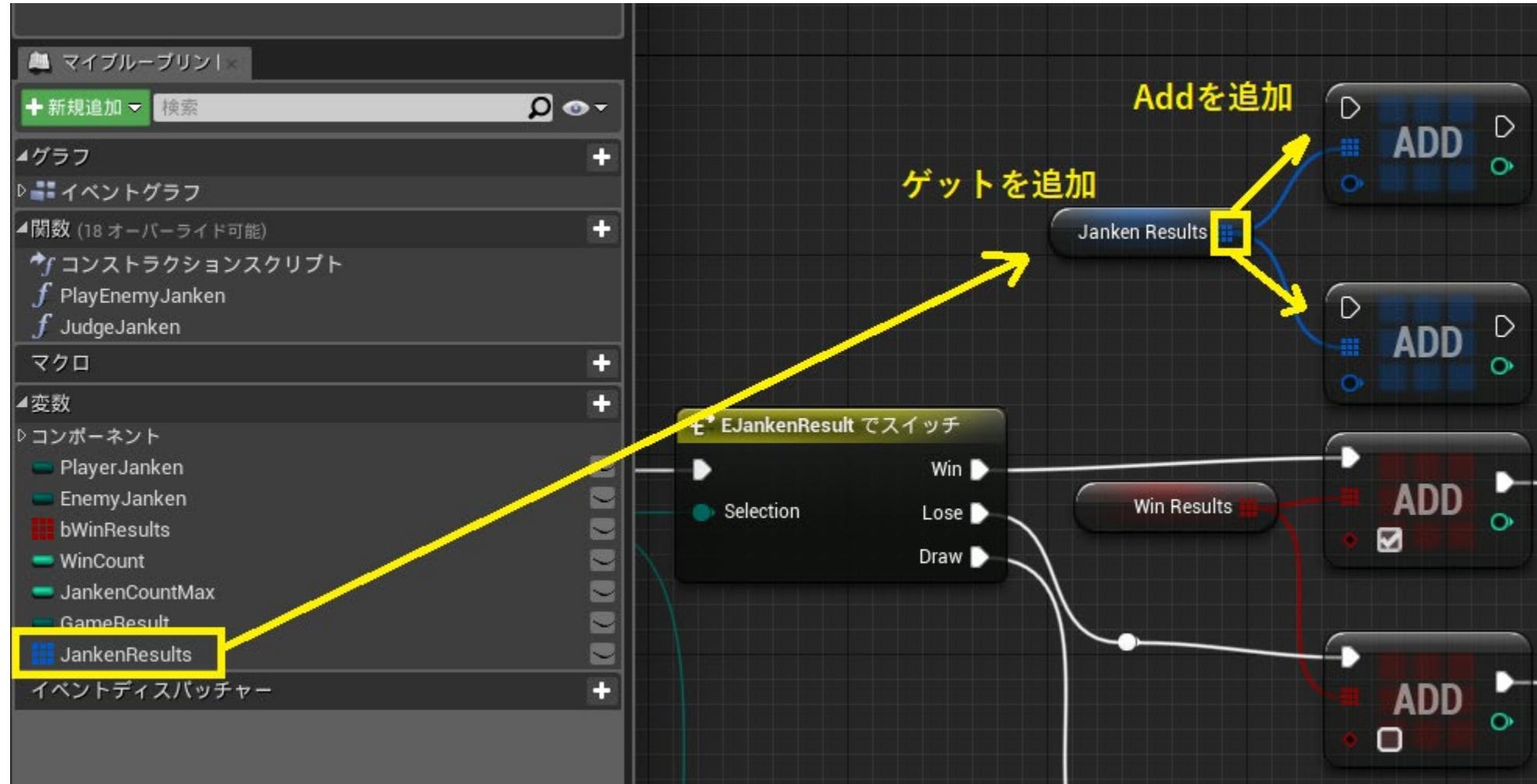
bWinResultを使用している箇所をFJankenResultsに置き換える



変数の設定を構造体に変更



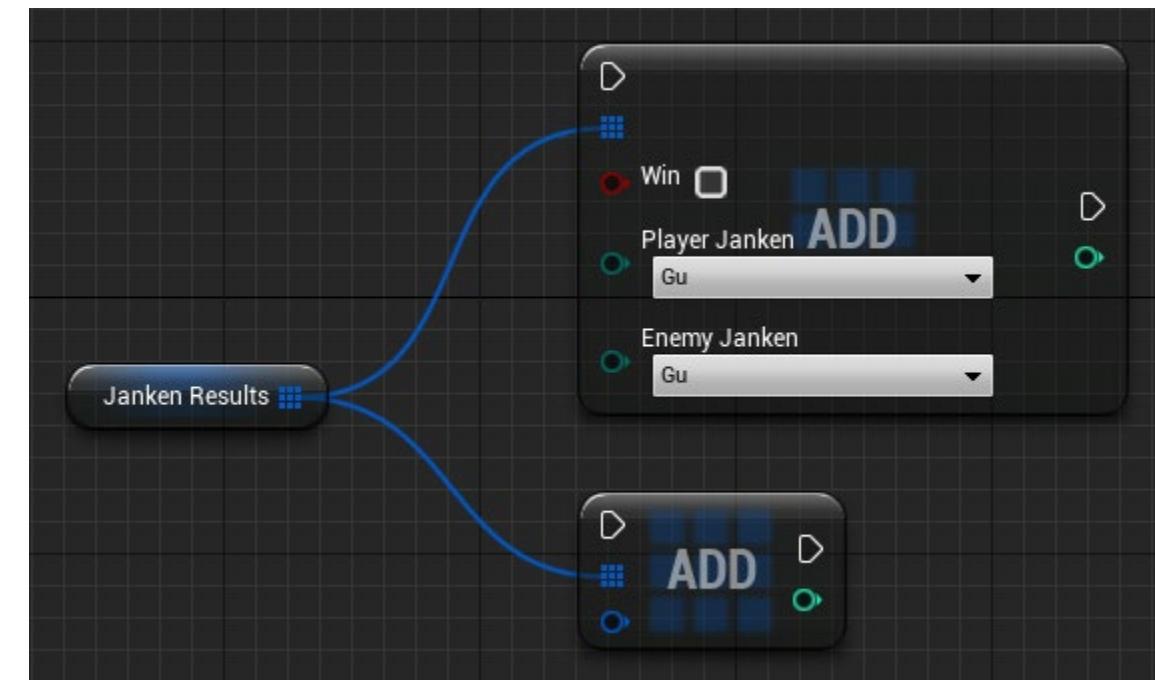
配列JankenResultsのゲットを追加して、
配列のAddを追加する



構造体ピンを分割

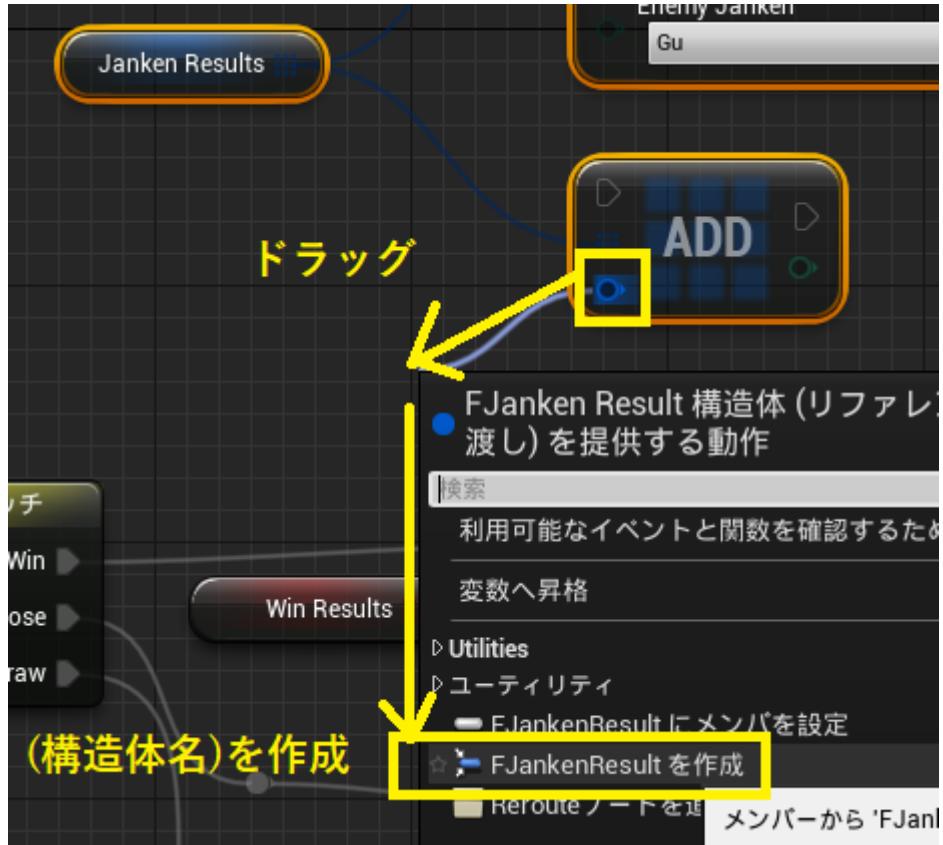


構造体のピンを右クリック
> 構造体ピンを分割

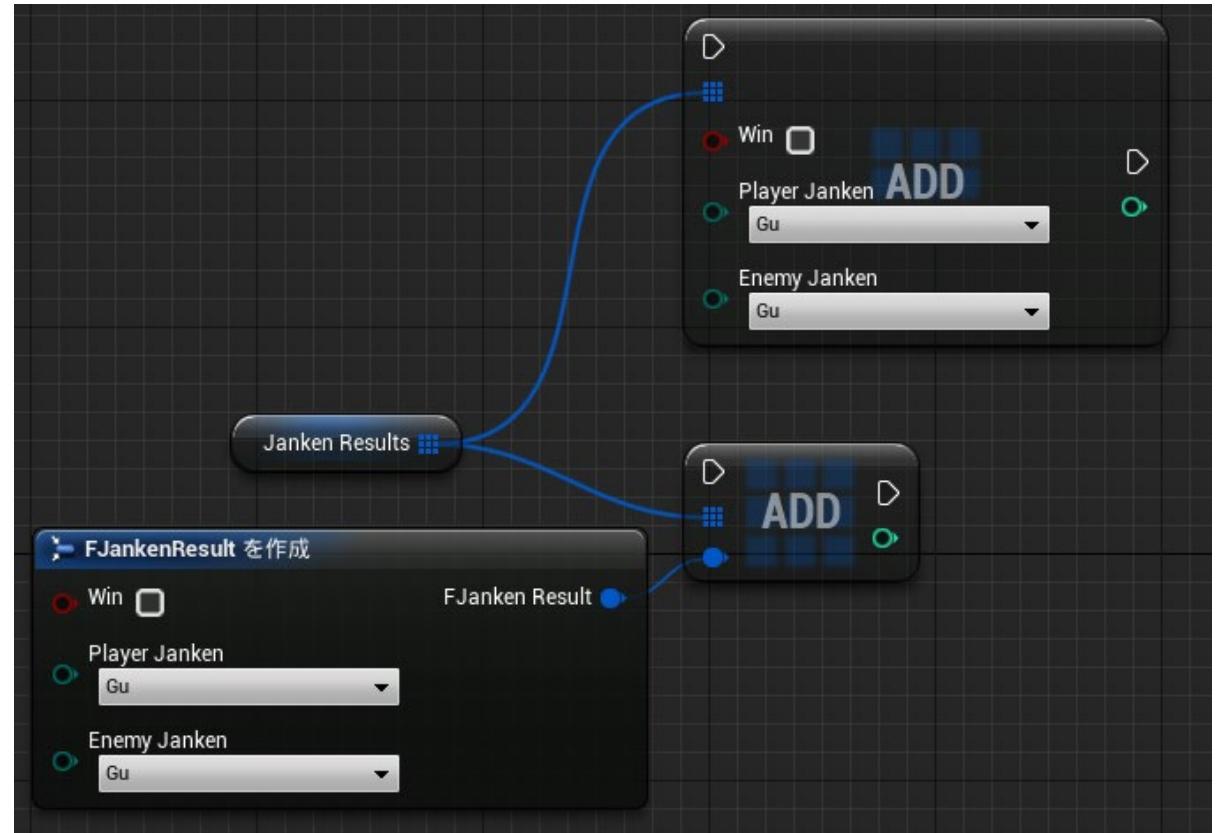


構造体のピンが変数ごとに設定できるようになる

構造体を作成

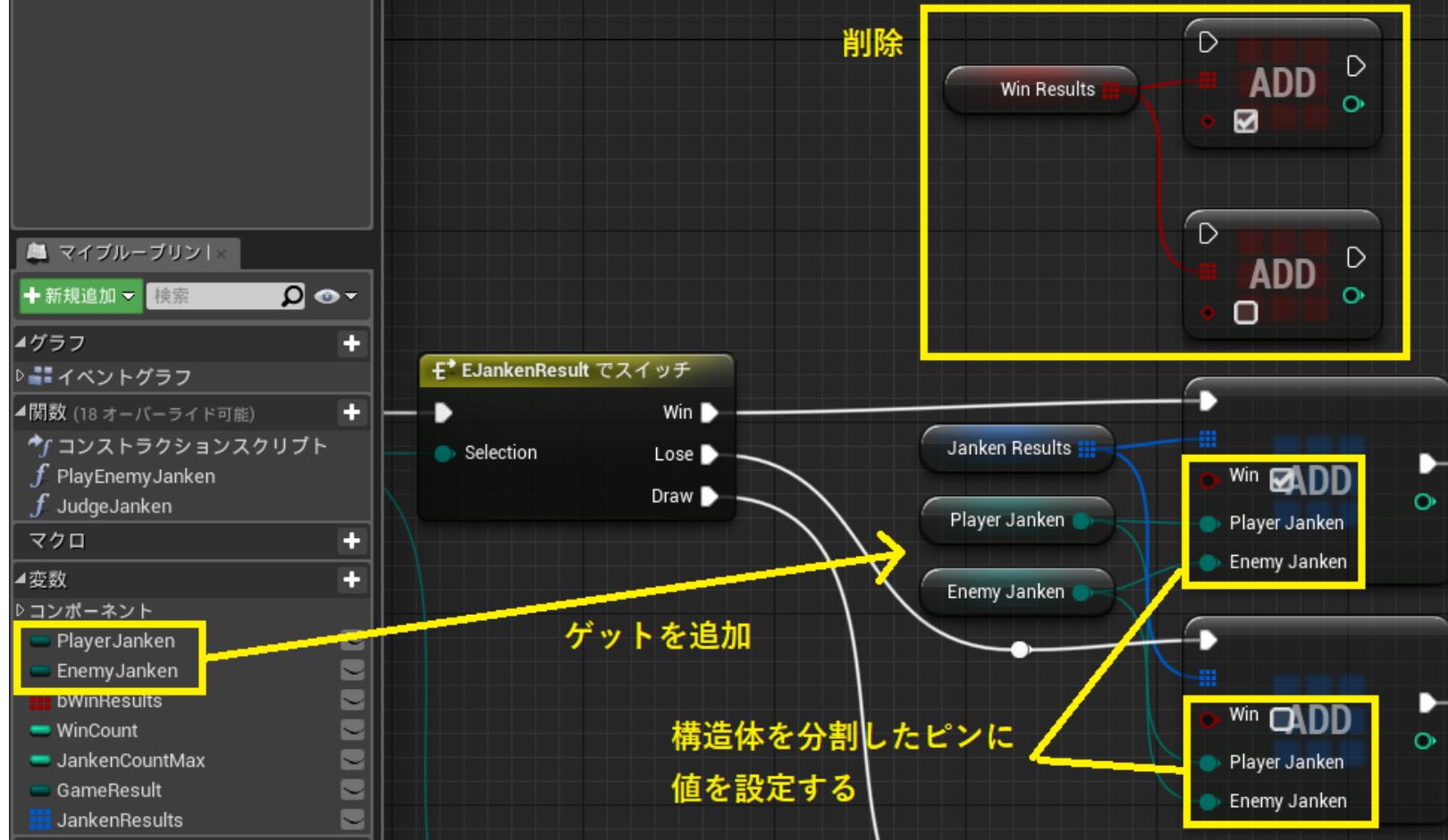


構造体のピンをドラッグ
> (構造体名)を作成



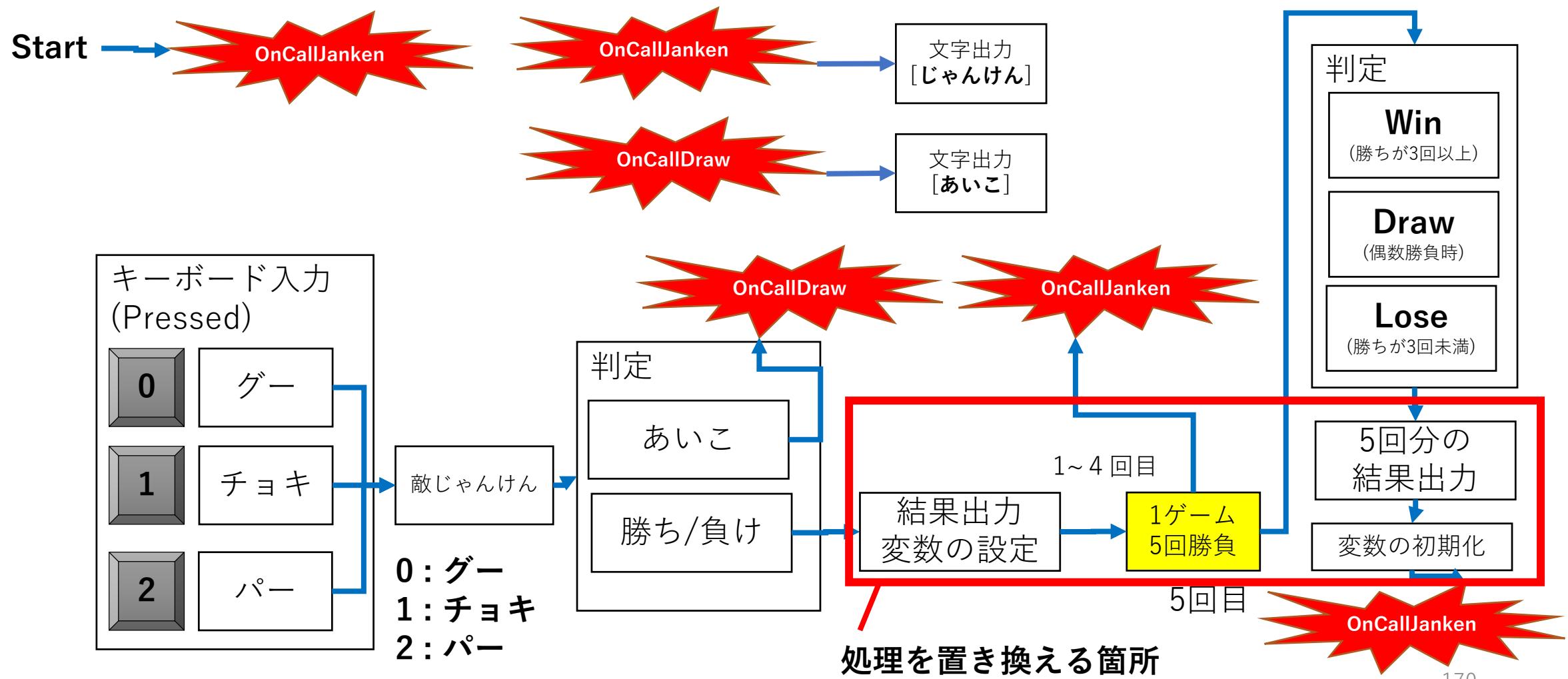
構造体を作成しても、構造体ピンを分割しても同じ
(構造体ピンを分割が後から増えた)

構造体を分割したピンに値を設定する WinResultsの処理と置き換える

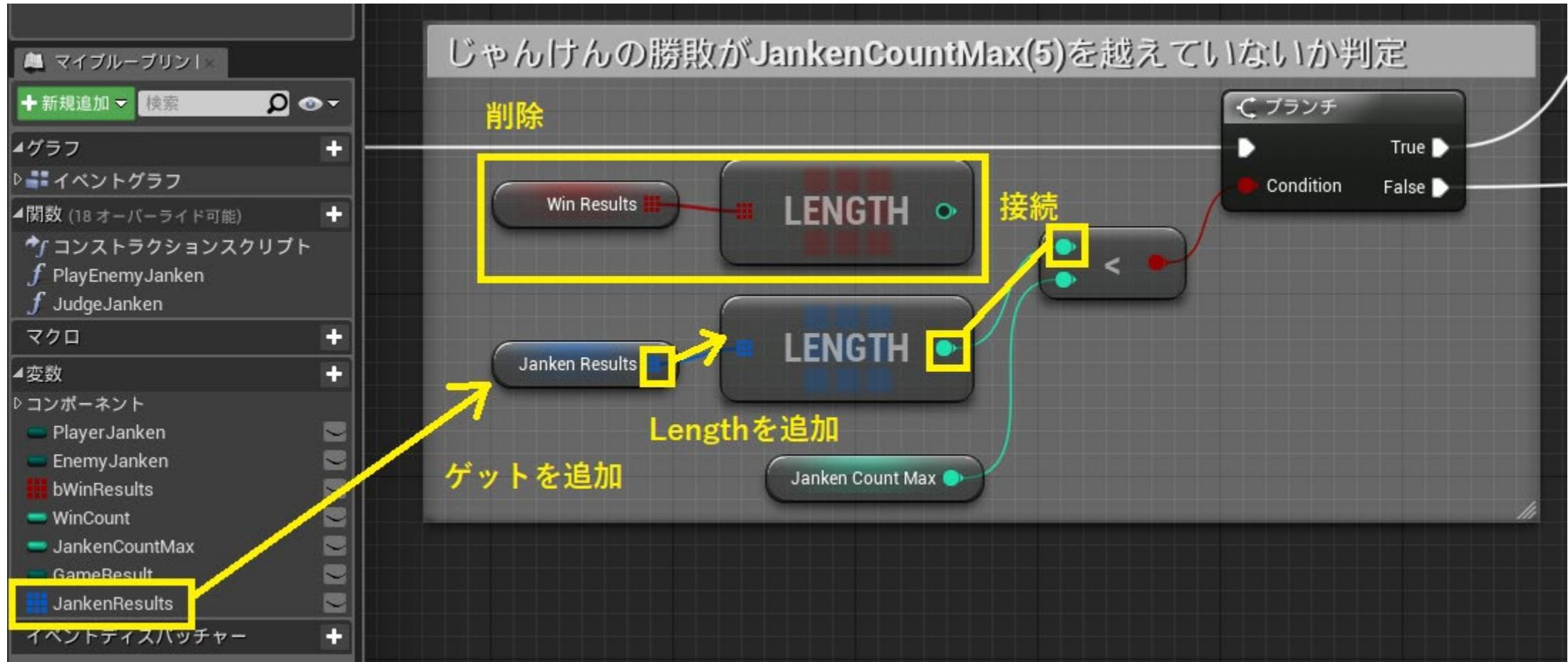


PlayerJanken, EnemyJanken のゲットを追加し、構造体を分割したピンに値を設定する
WinResults の Add 処理を置き換え、不要になった処理を削除する

変数の設定を構造体に変更

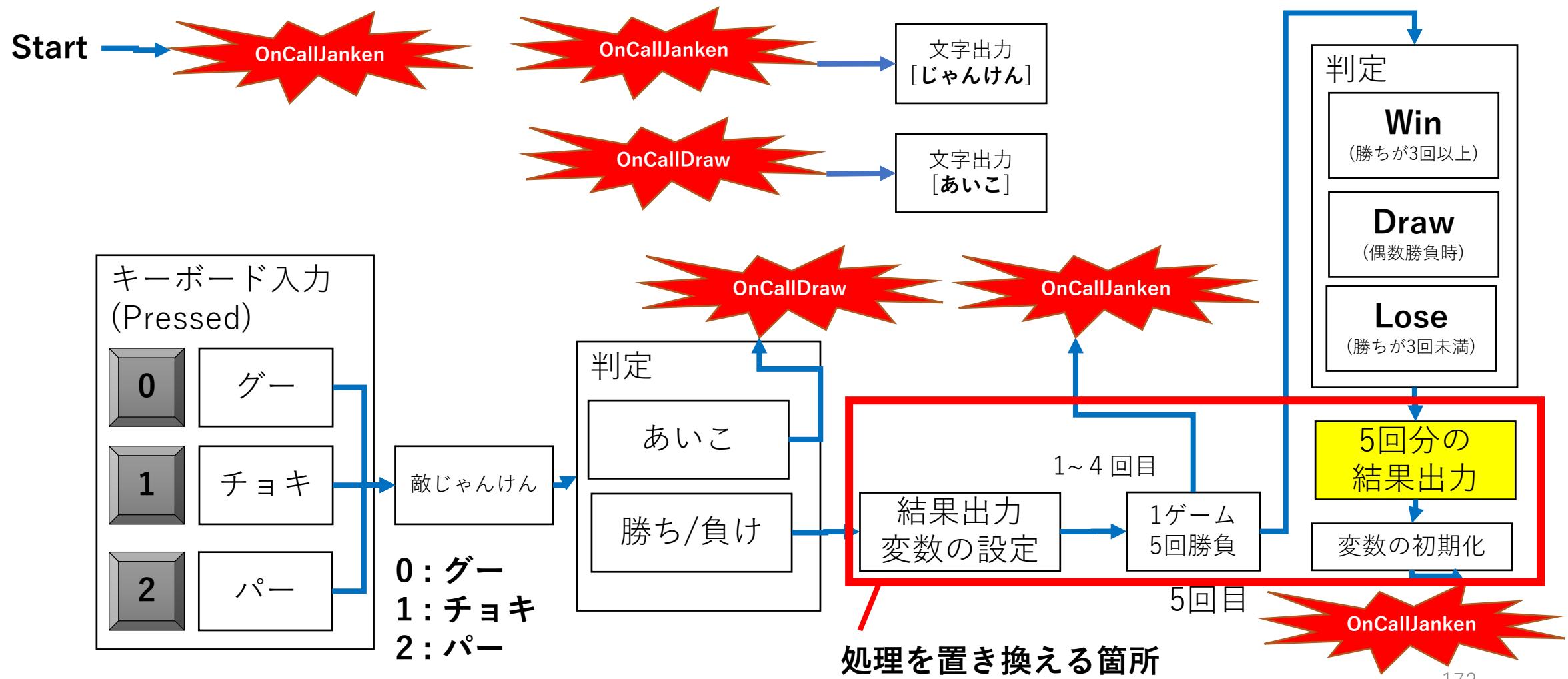


配列のLengthを接続している箇所を置き換える

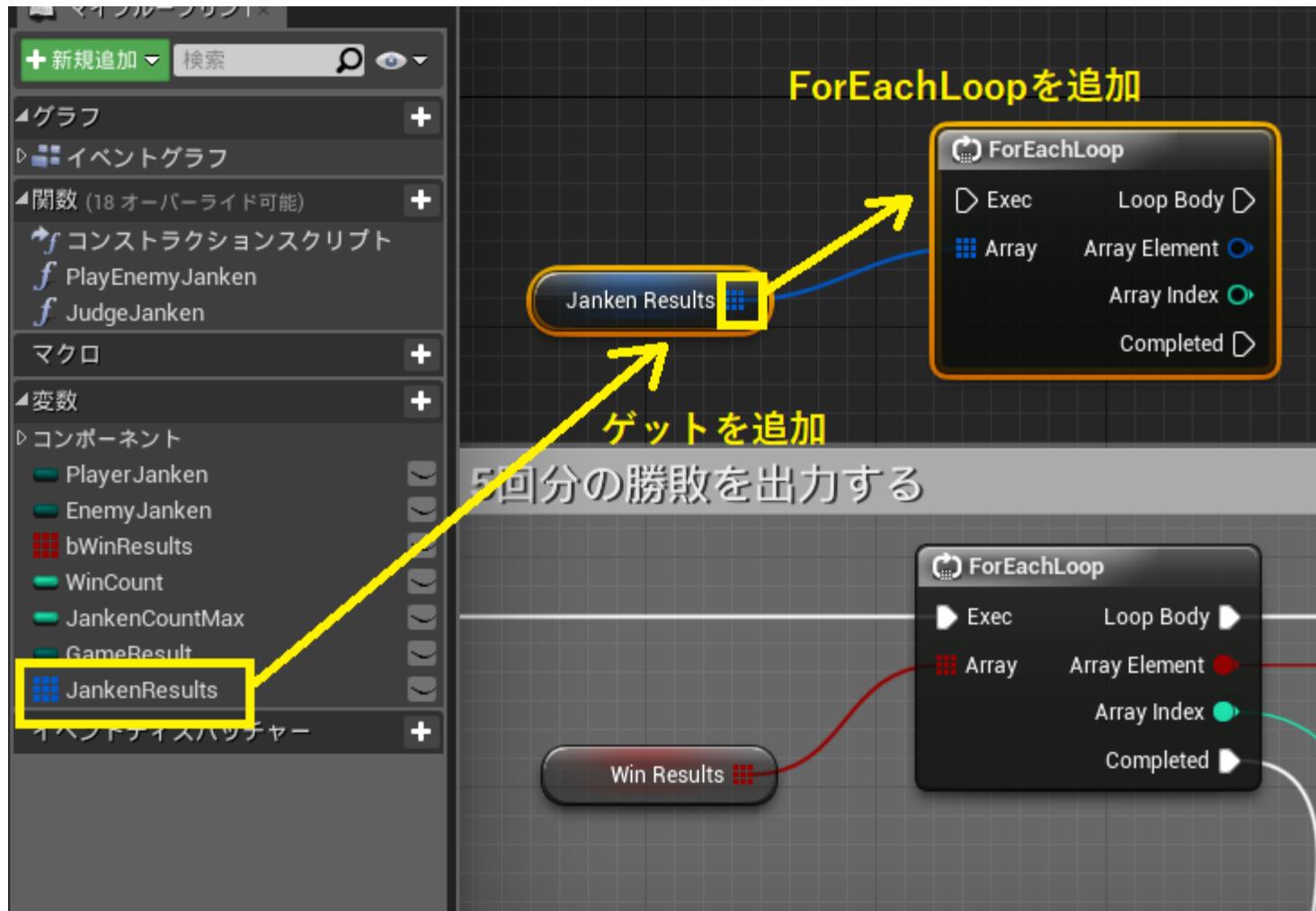


JankenResultsのゲットを追加 > Lengthを追加 > Lengthの接続をつなぎ変える
不要になったWinResultsの処理は削除

5回分の結果出力を構造体に変更

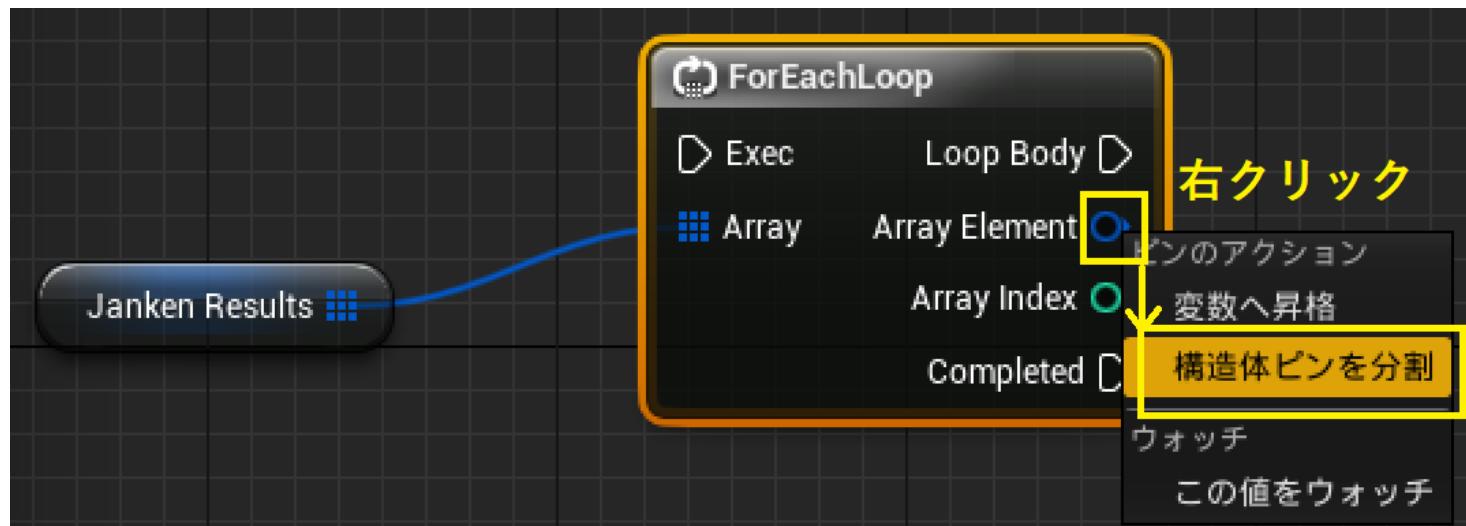


配列のForEachLoopを追加

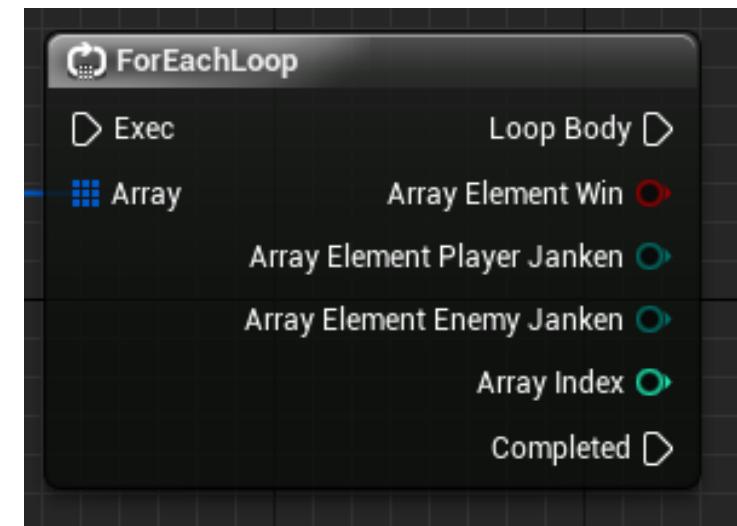


JankenResultsのゲットを追加
> ForEachLoopを追加

構造体ピンを分割

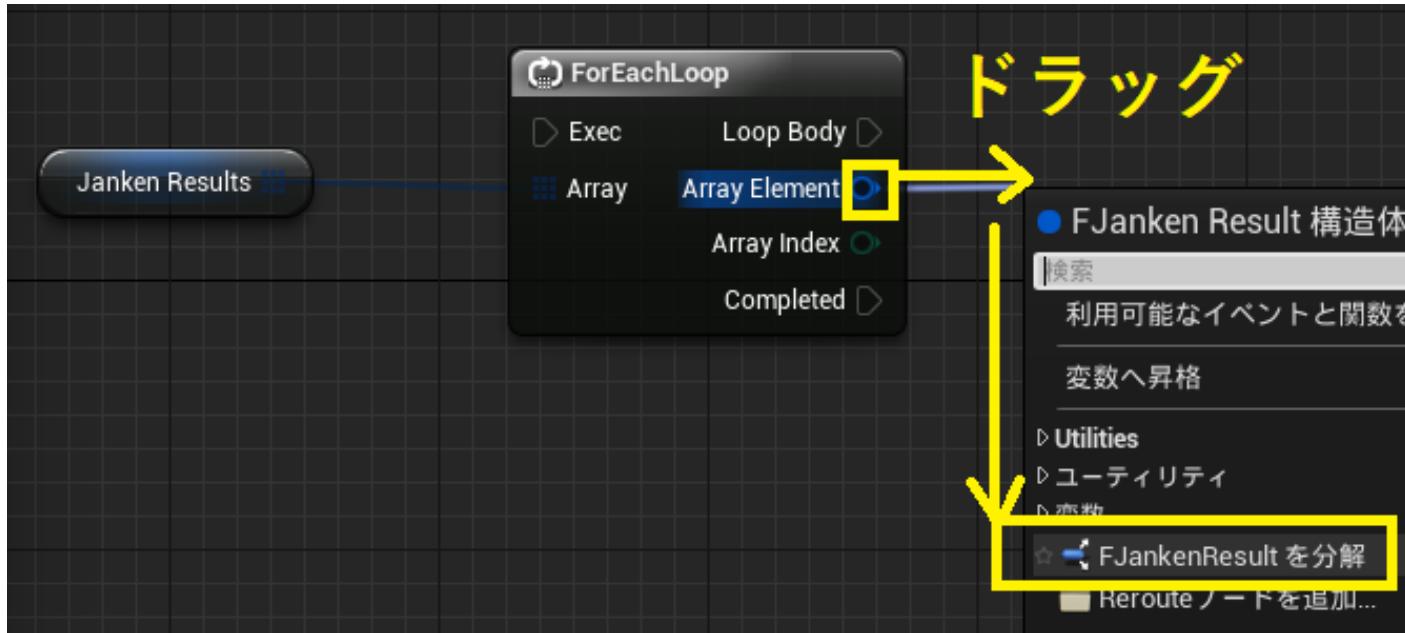


Array Elementを右クリック > 構造体ピンを分割

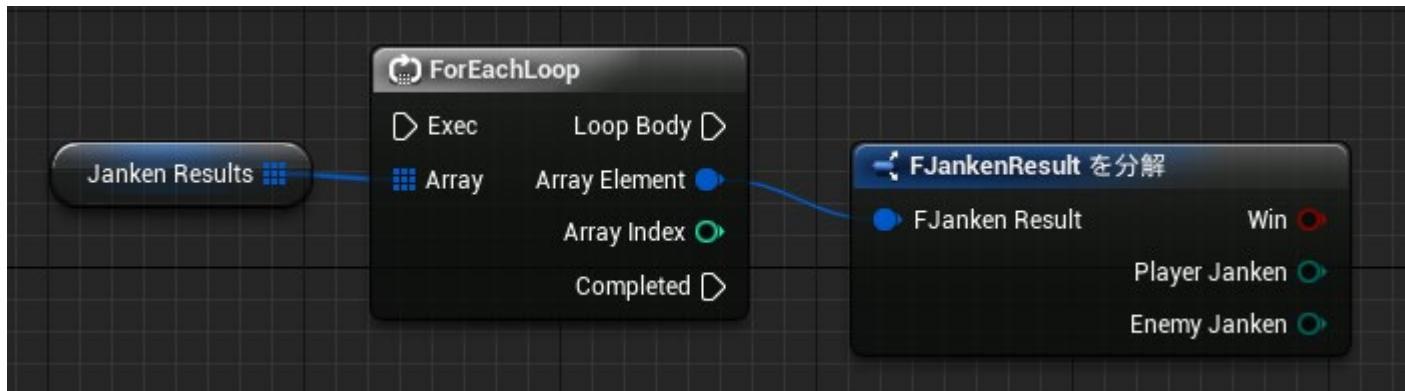


Array Elementが変数単位に分割される

構造体を分解

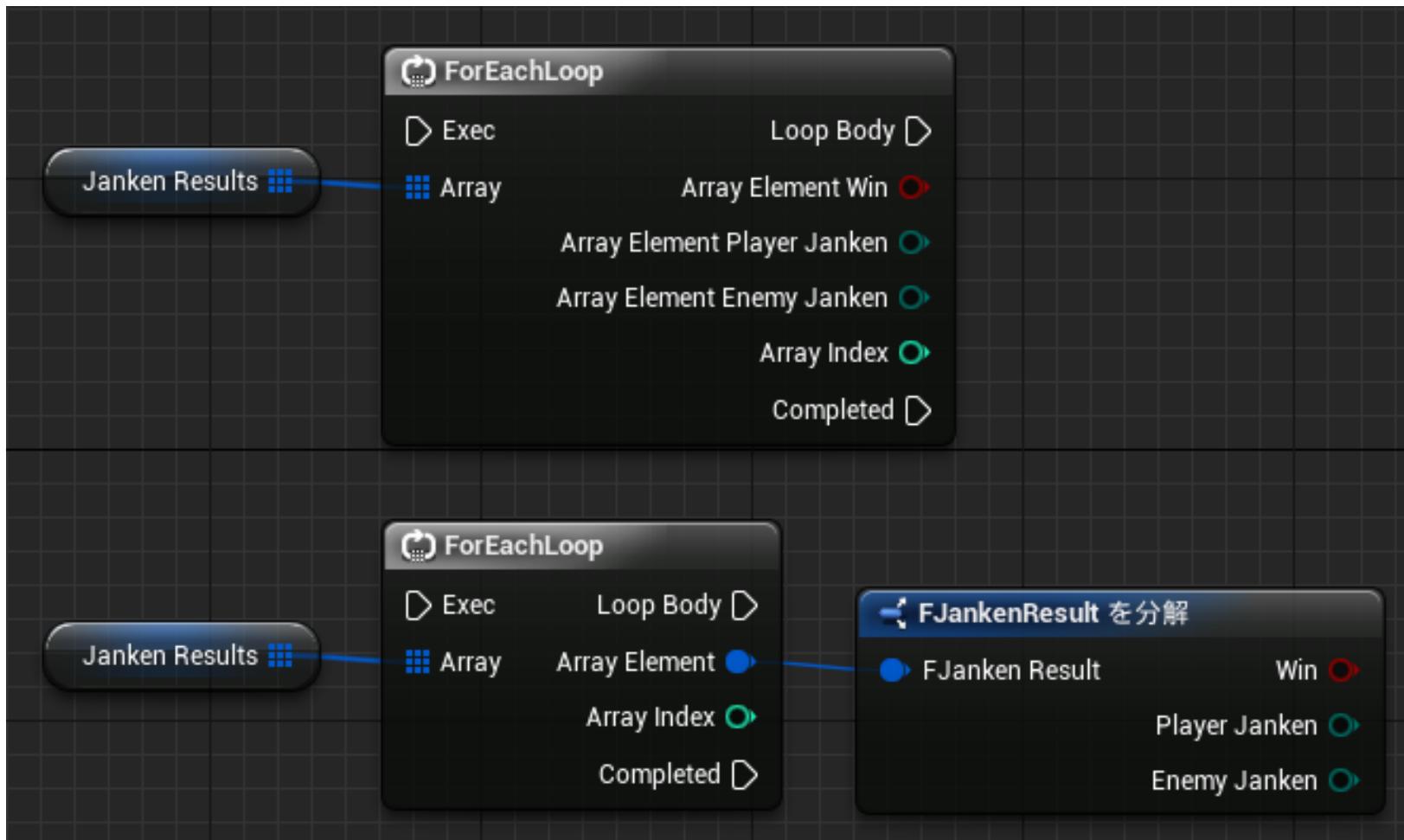


Array Elementをドラッグ
> (構造体名)ピンを分解



(構造体名)ピンを分解のノードが作成される

1つのノードか、分解ノードを作るか

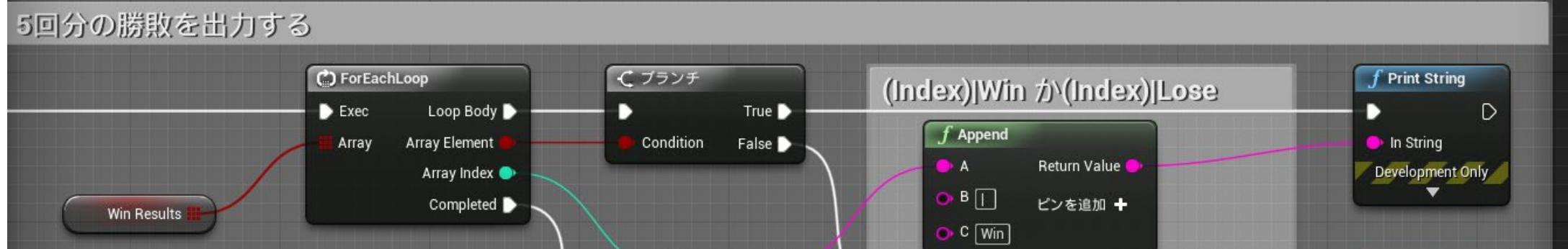
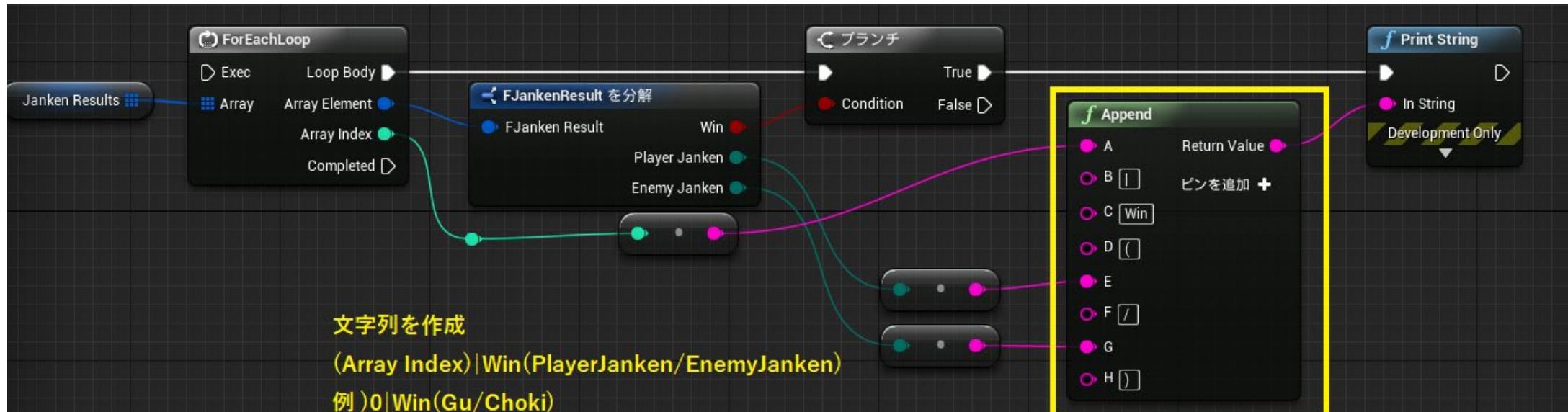


どちらにするかは好み
セットの時はピンを分解した方が
設定しやすい

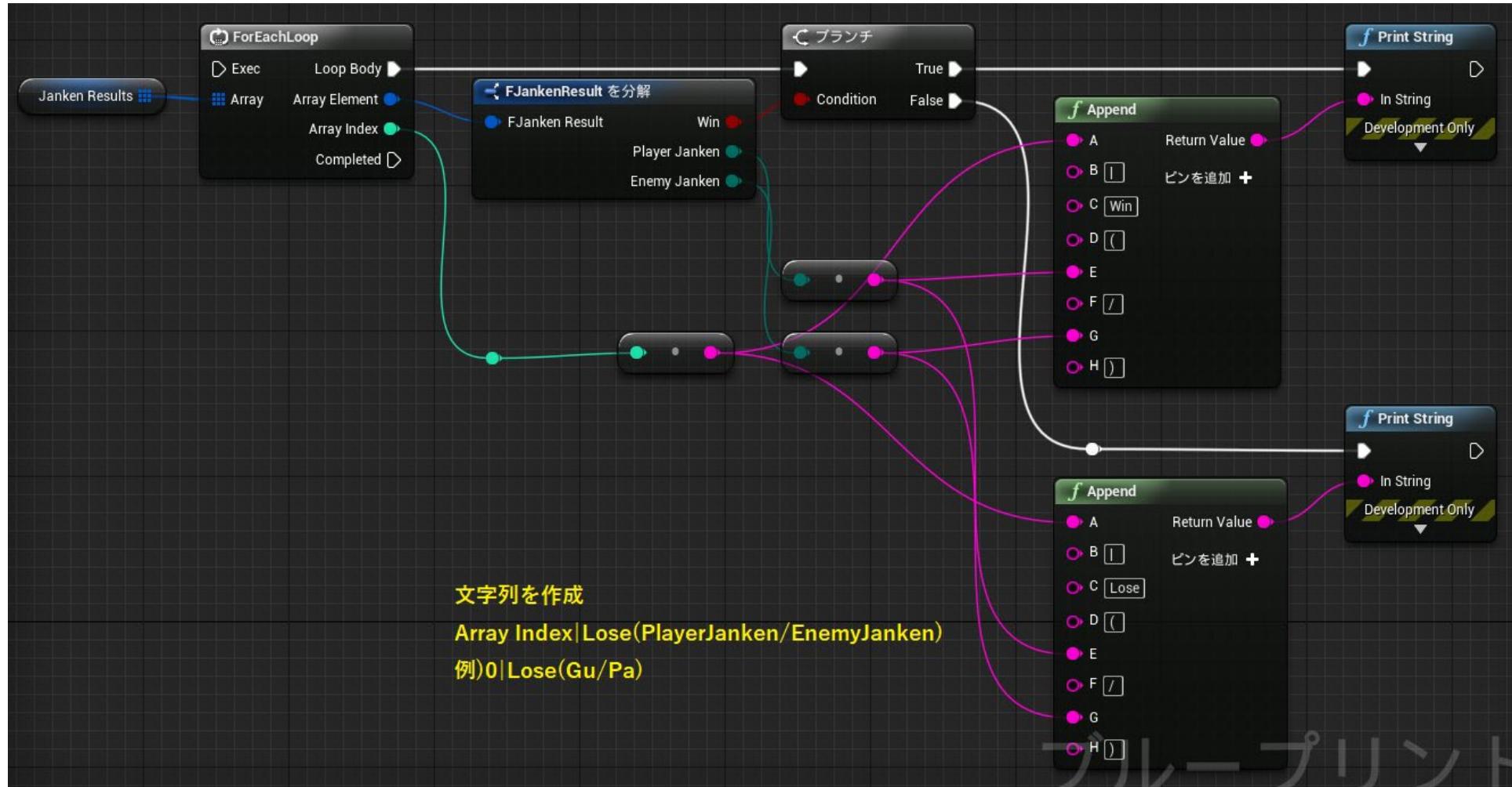
ゲットの時は分解ノードを作った方が分かりやすい

例) 関数のアウトプットで複数の構造体を返すような場合にピンを分割してしまうと、どの構造体の変数か
わからなくなってしまう

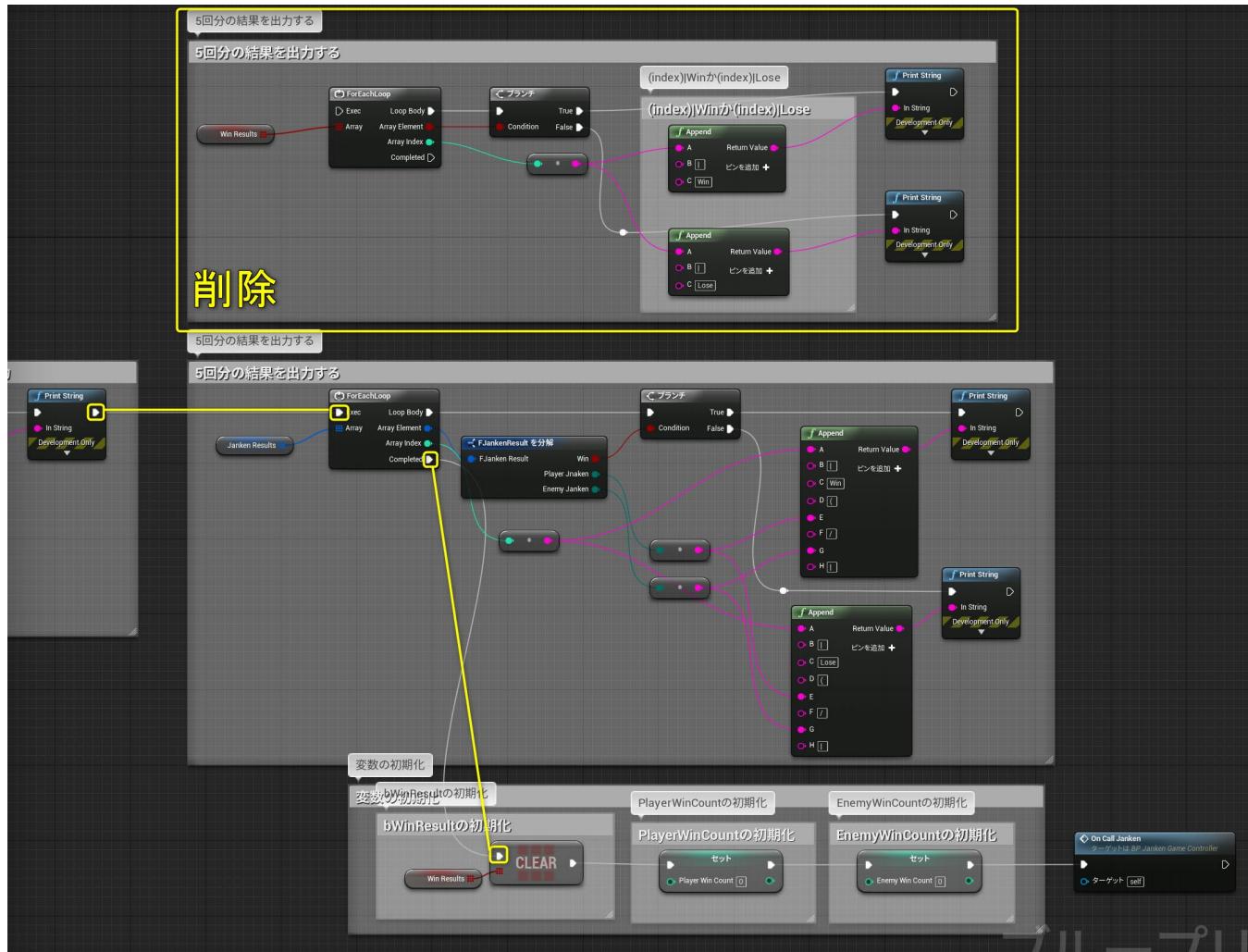
勝ちの場合の文字列を作成して出力



負けの場合の文字列を作成して出力



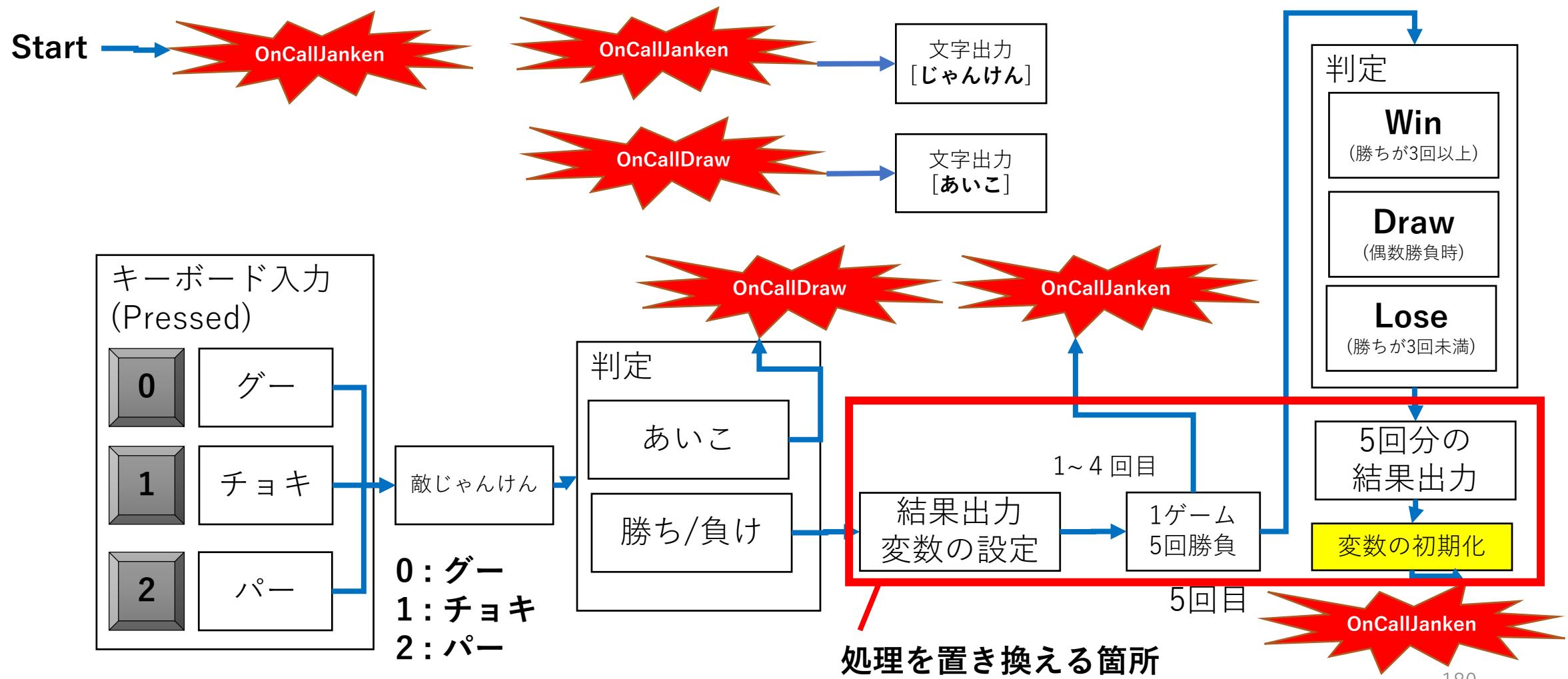
bWinResultの出力処理をJankenResultsの処理に置き換える
bWinResultの出力処理は削除する



bWinResultの出力処理から、
JankenResultsの処理に置き換える

bWinResultの出力処理は使用しないので削除する

変数の設定を構造体に変更

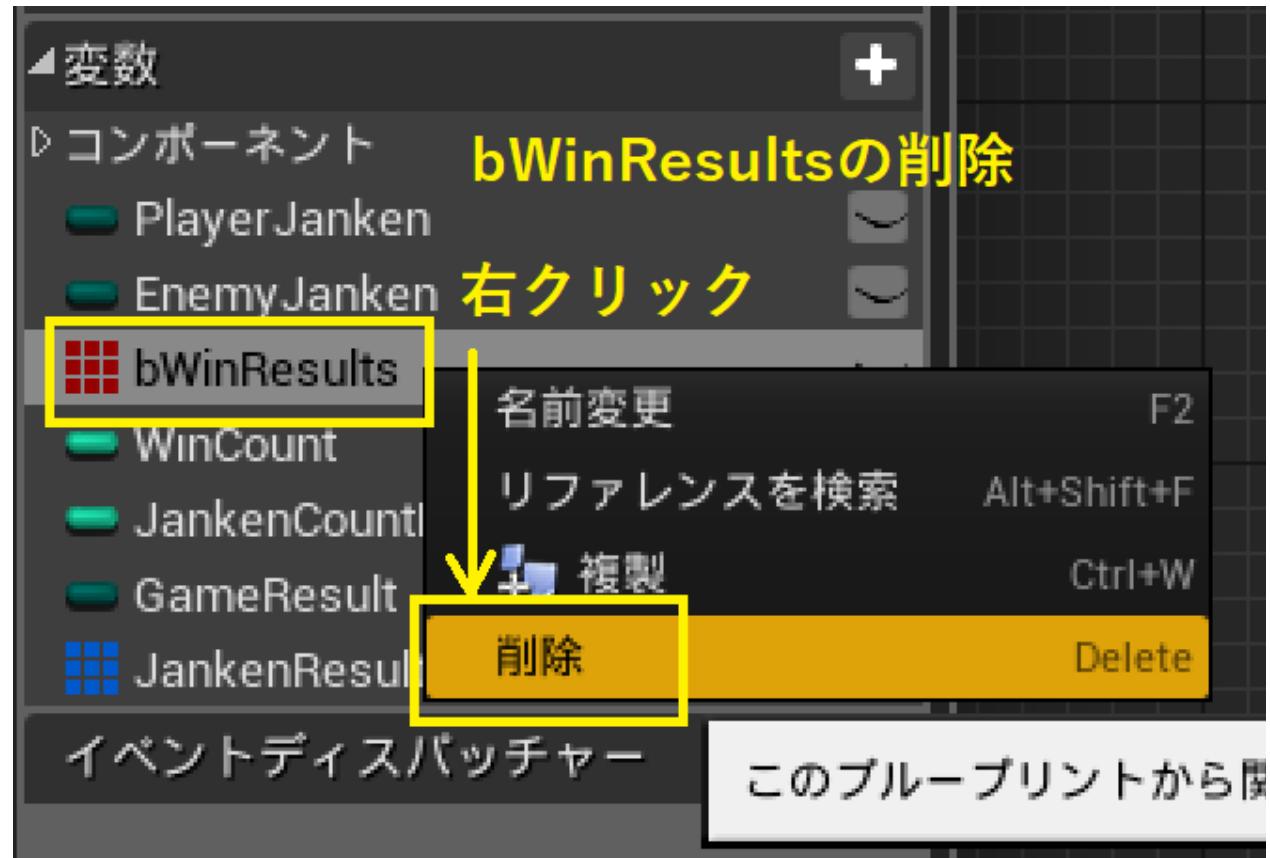


配列のクリア処理を置き換える



JankenResultsのGetを追加 > Clearを追加 > JankenResultsのClearを呼ぶように接続する
bWinResultsの処理は削除する

bWinResultsを削除し、コンパイルにエラーが無ければ置き換え完了

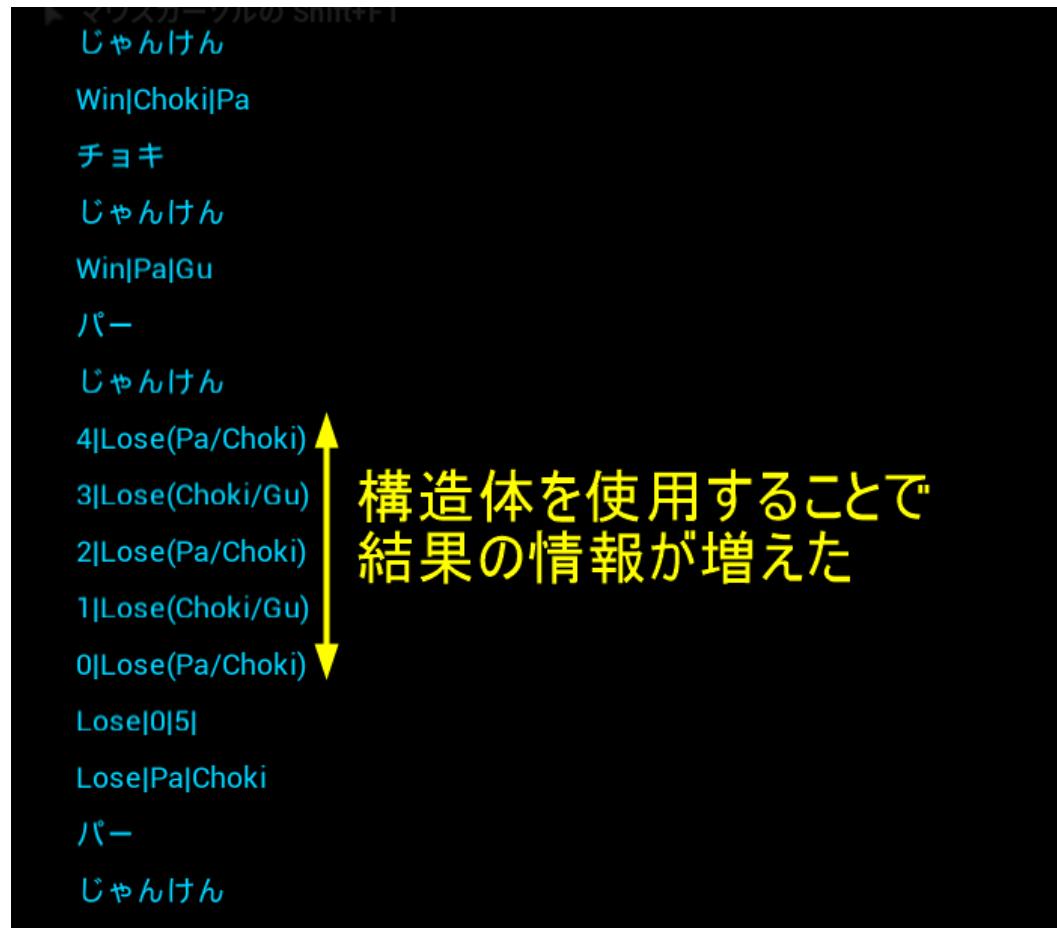


使用していないのでbWinResultsは削除する



コンパイルを実行して、
エラーが無ければ完了

プレイして確認する



7.じゃんけんゲームのUI作成

7.じゃんけんゲームのUI作成

7.1 UIに使用する画像のインポート・設定

7.2 ウィジェットブループリントの操作のおさらい

7.3 ゲームのじゃんけん選択画面を作成する

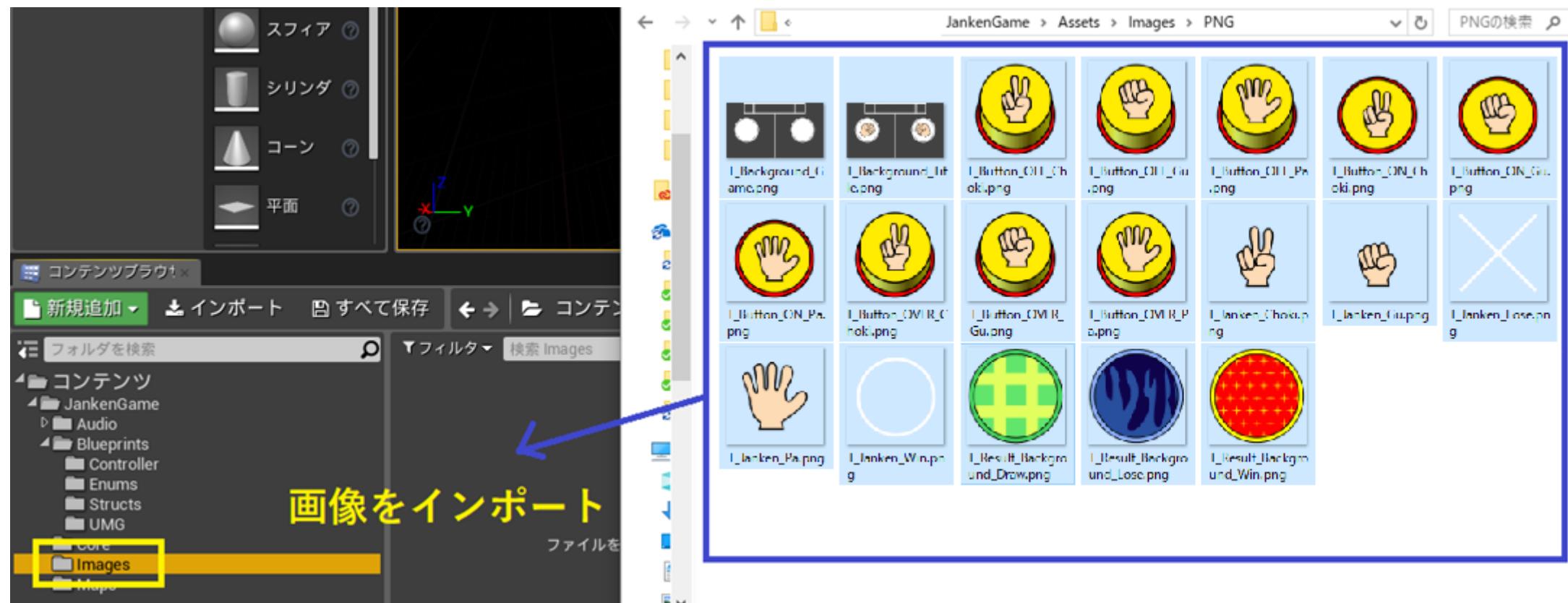
7.じゃんけんゲームのUI作成

7.1 UIに使用する画像のインポート・設定

7.2 ウィジェットブループリントの操作のおさらい

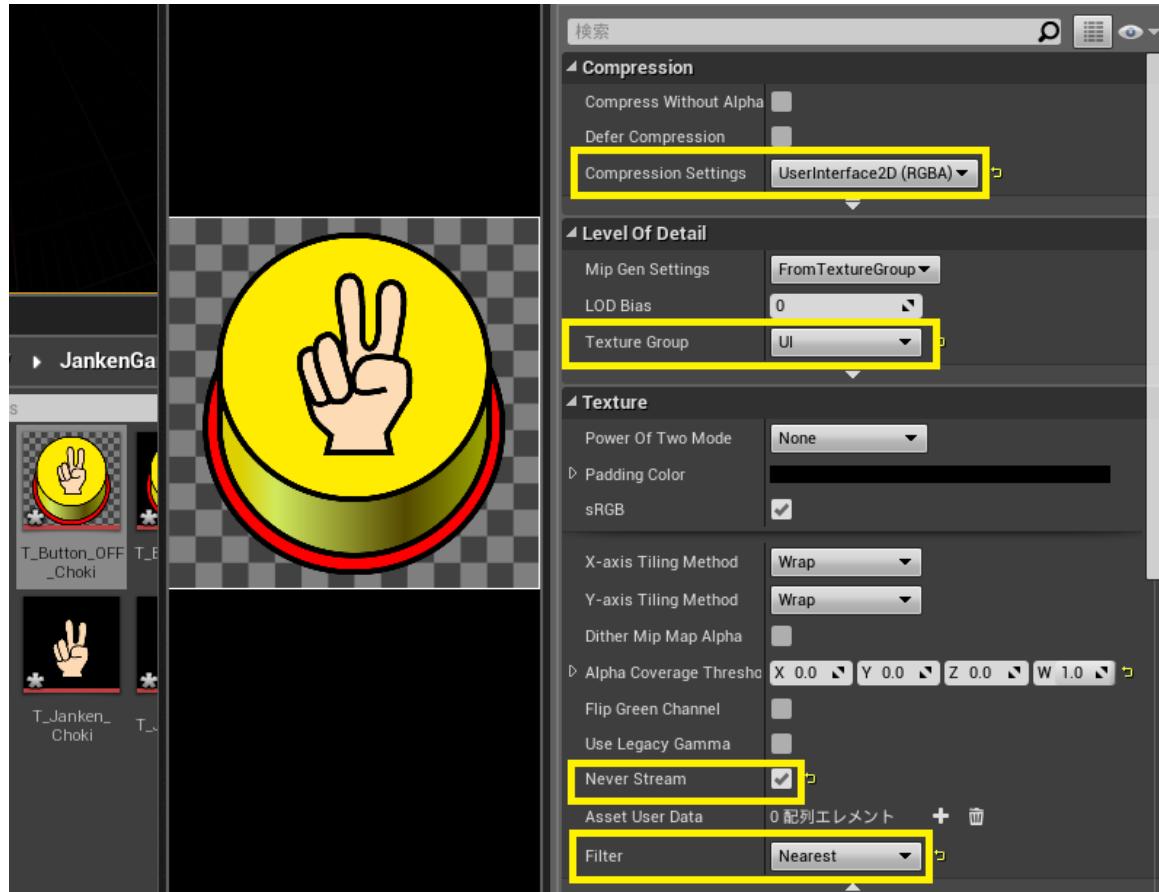
7.3 ゲームのじゃんけん選択画面を作成する

画像のインポート



Assets > Images > PNG配下の画像をImagesフォルダに画像をインポートする

画像の設定をUI用に設定する



画像をダブルクリックで設定ウィンドウを開き、
UI用の設定に変更する

項目	設定値
Compression Settings	UserInterface2D
Texture Group	UI
Never Stream	チェック
Filter	Nearest



設定を行うと透過が表示される

すべての画像に設定を行う



全ての画像に設定を行う

UE4 UI用画像の設定について

http://denshikousakubu.com/2017/10/24/20171024_UE4_UMGTextureSettings/

7.じゃんけんゲームのUI作成

7.1 UIに使用する画像のインポート・設定

7.2 ウィジェットブループリントの操作のおさらい

7.3 ゲームのじゃんけん選択画面を作成する

ウィジェットブループリントの作成

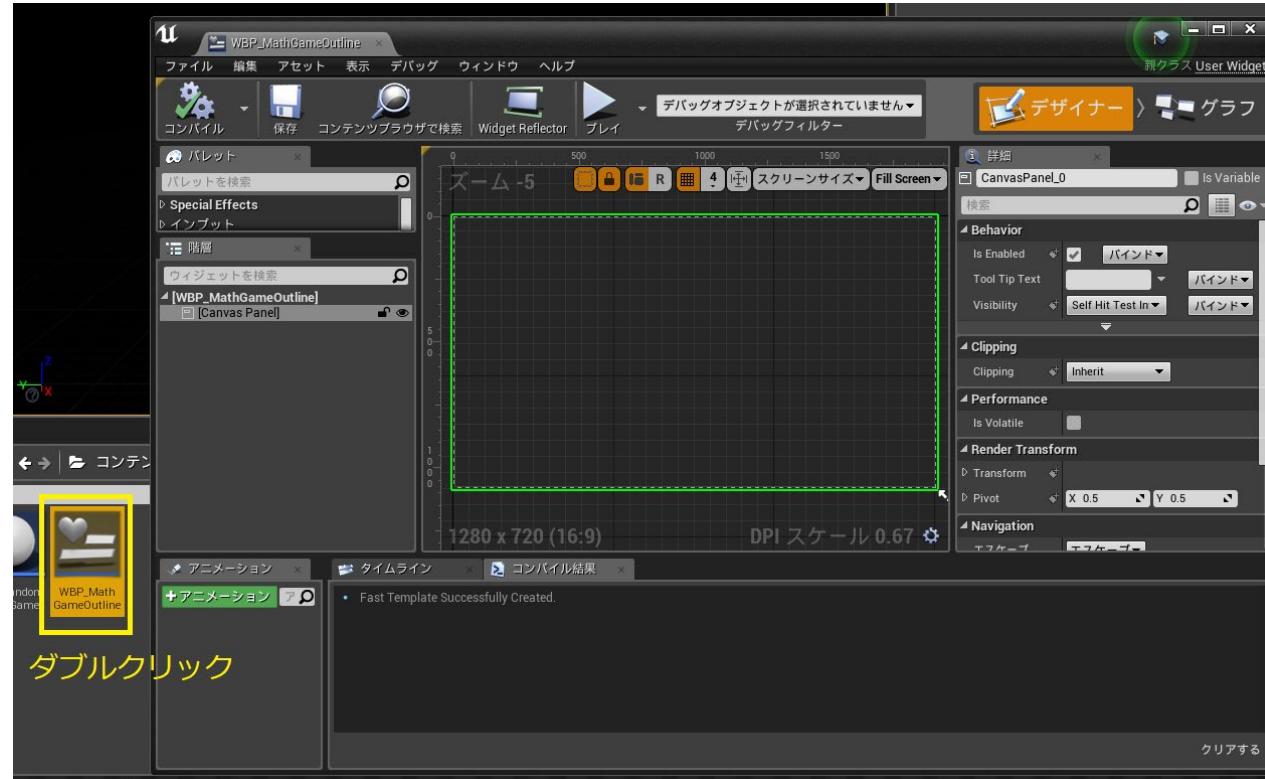


Blueprints フォルダを選択
右クリック > ユーザーインターフェース
> ウィジェットブループリント



名前を設定する

ウィジェットブループリントエディタについて



ウィジェットブループリントエディタの詳細

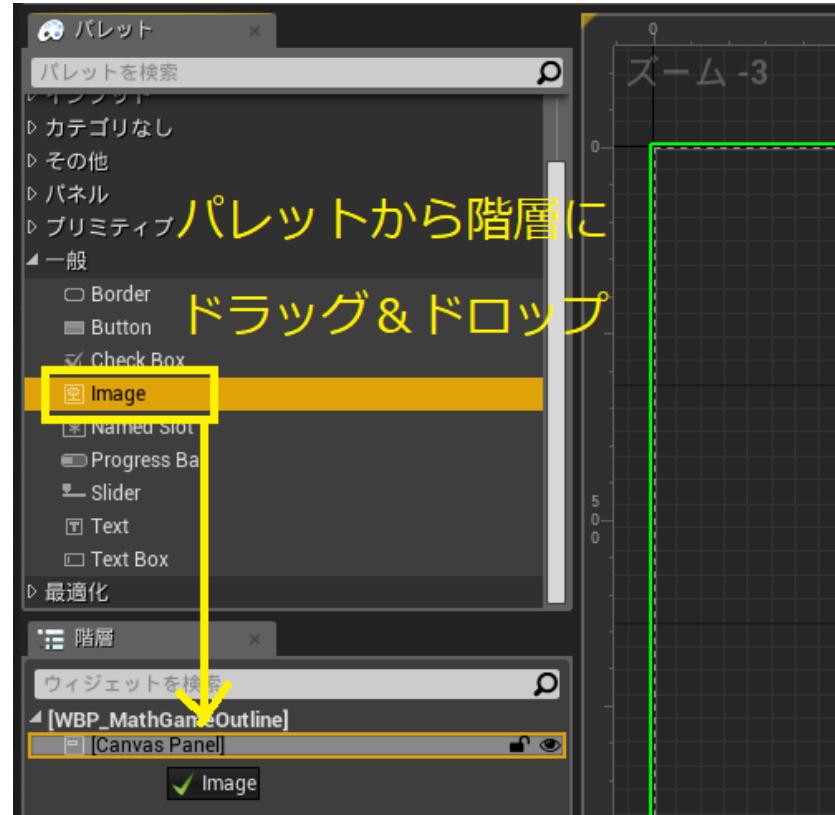
<https://docs.unrealengine.com/latest/JPN/Engine/UMG/UserGuide/WidgetBlueprints/index.html>



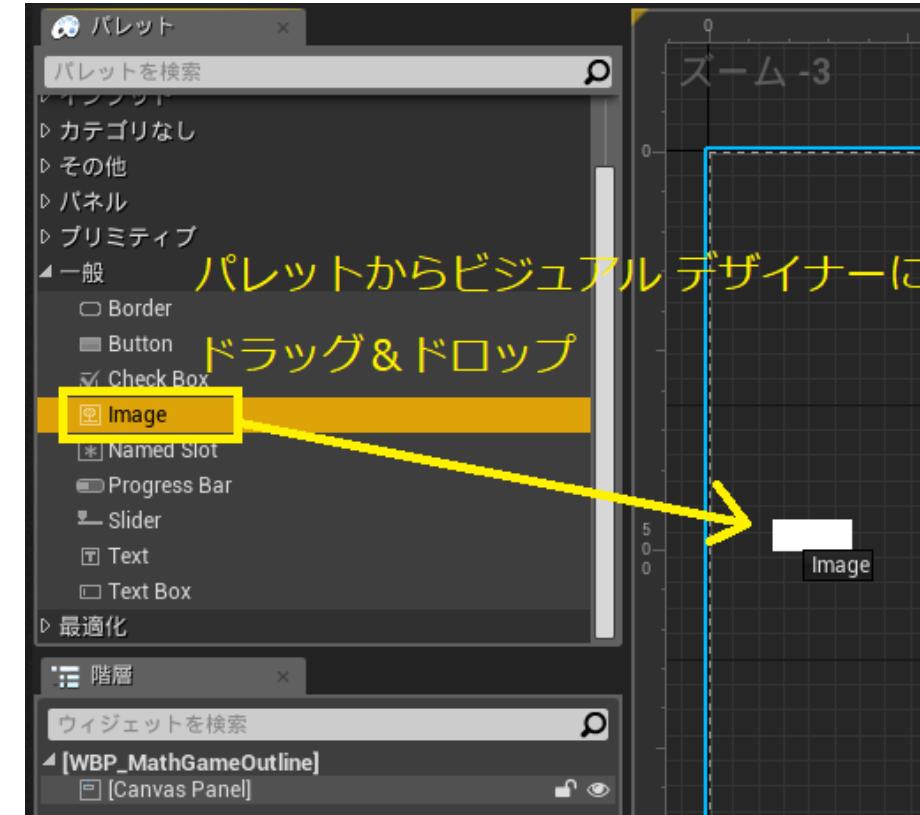
デザイナー
UIを作成

グラフ
UIの処理を作成

ウィジェットの追加

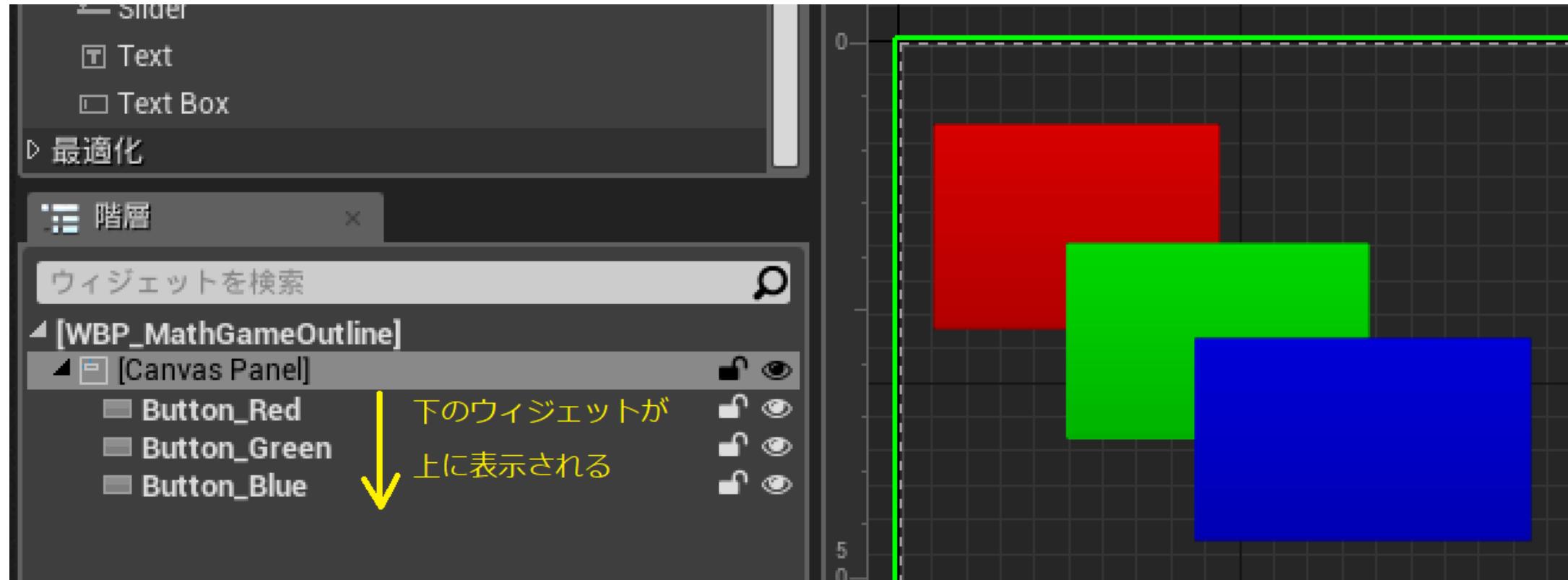


パレットからウィジェットを
階層タブにドラッグ&ドロップ



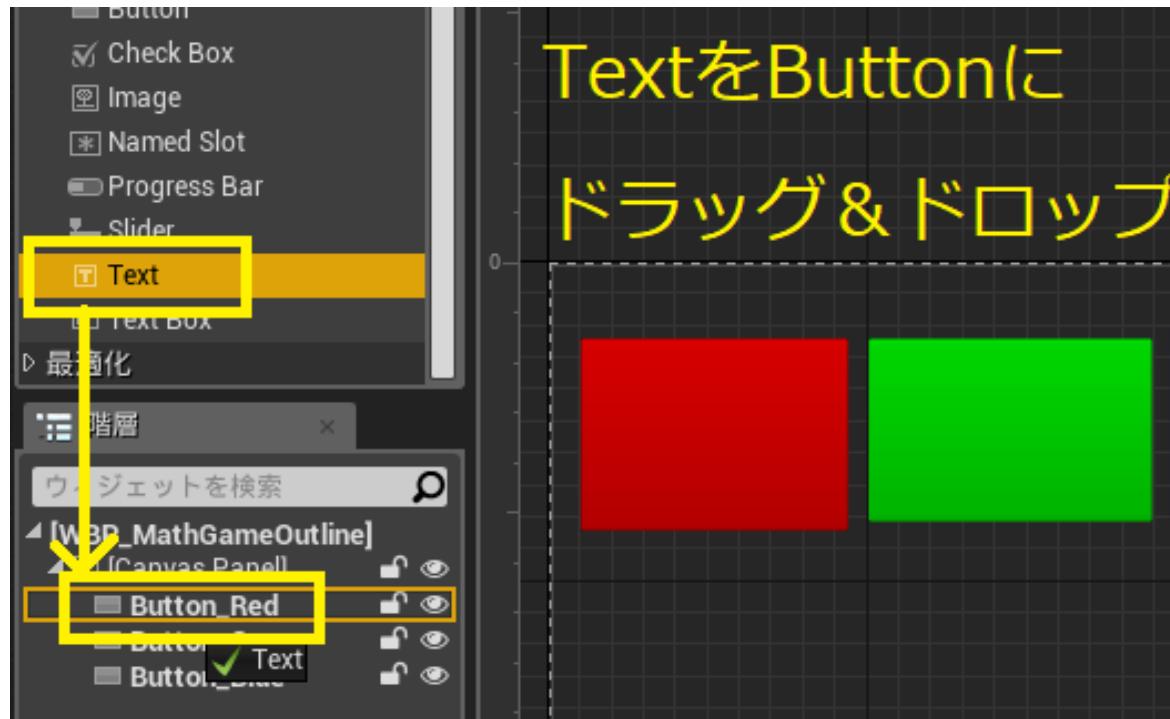
パレットからウィジェットを
ビジュアルデザイナーにドラッグ&ドロップ

階層タブで表示されるウィジェットの順番を設定

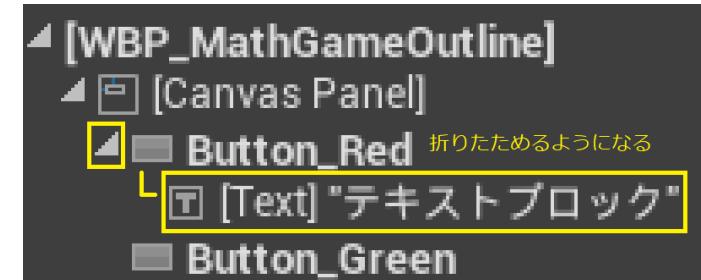


階層タブで表示される
ウィジェットの順番を決める
下のウィジェットが上に表示される

ウィジェットの親子関係(テキストを含むボタン)



TextをButtonにドラッグ&ドロップ
(ButtonにTextを子として追加すると、
Textのあるボタンを作成することができる)

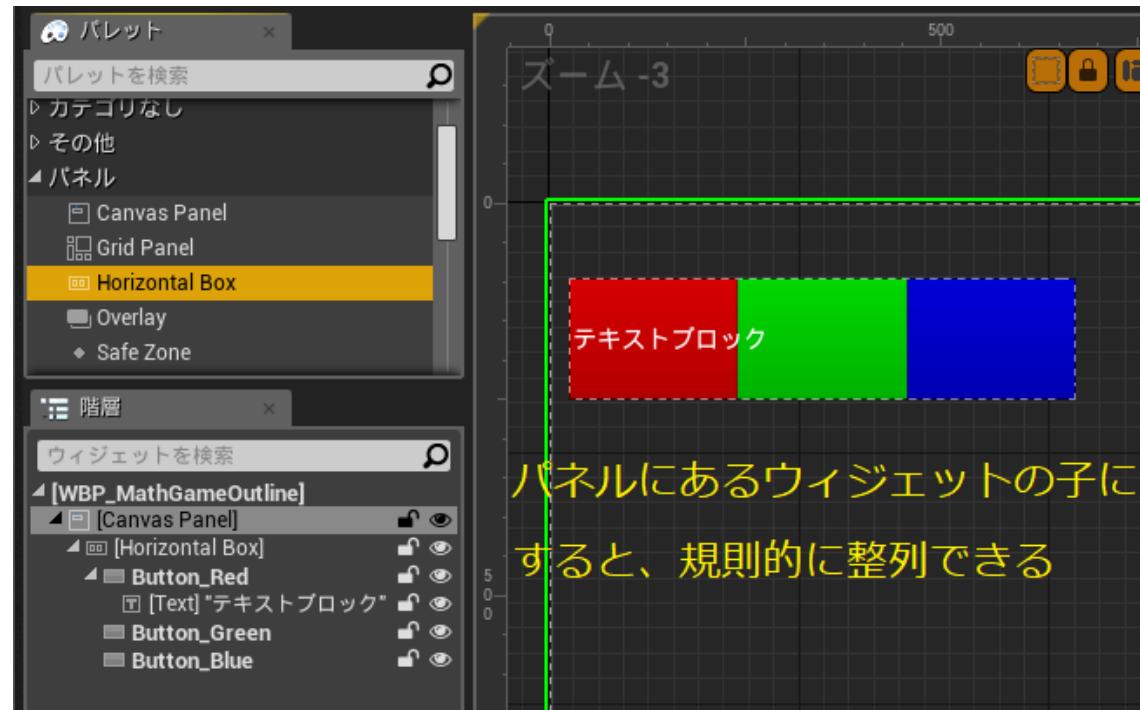


子が増えると折りたためる
ようになる

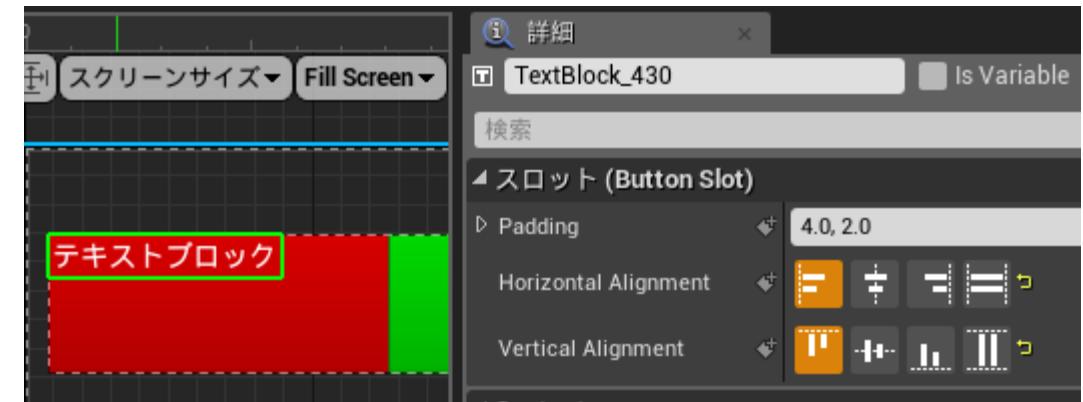


親のボタンを動かすと
子のテキストも一緒に動く

ウィジェットの子になると配置方法が変わる

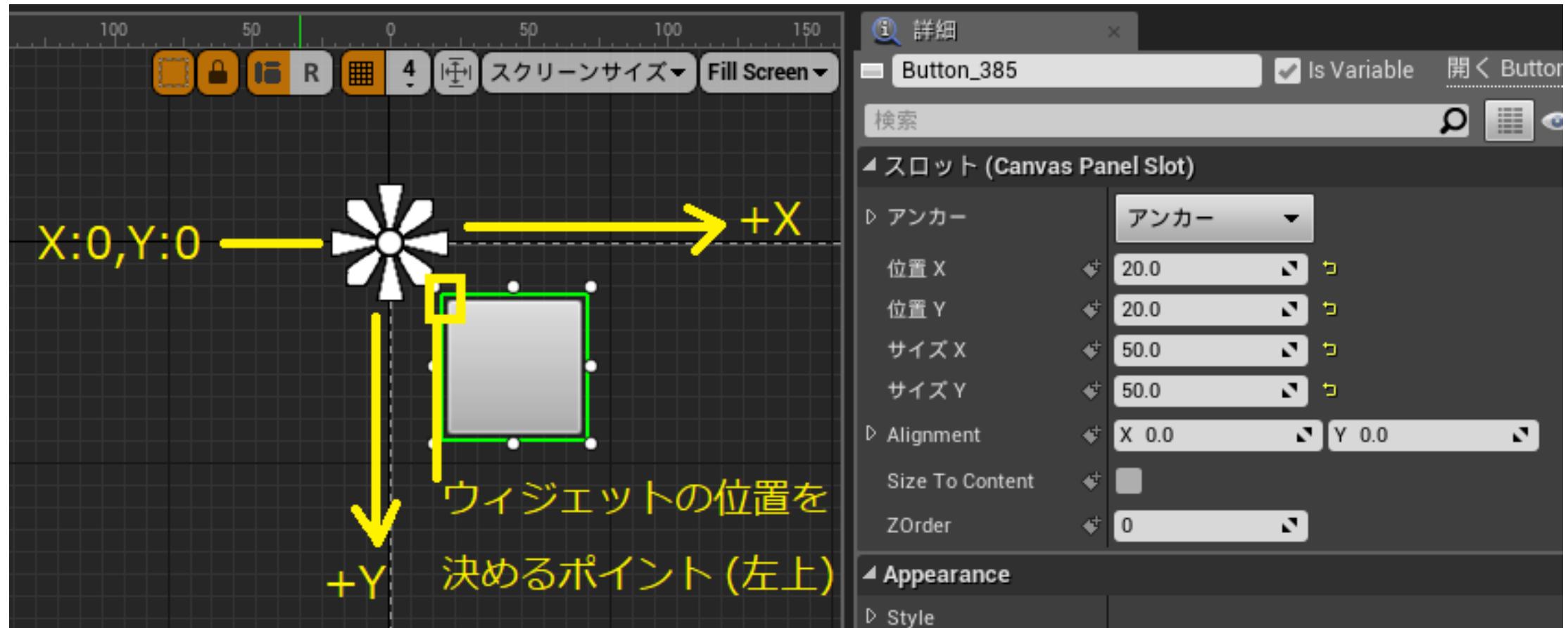


パネルにある項目の子供にすると、
横や縦に規則的に配置することができる
(Horizontal Box : 横方向に規則的に配置することができる)

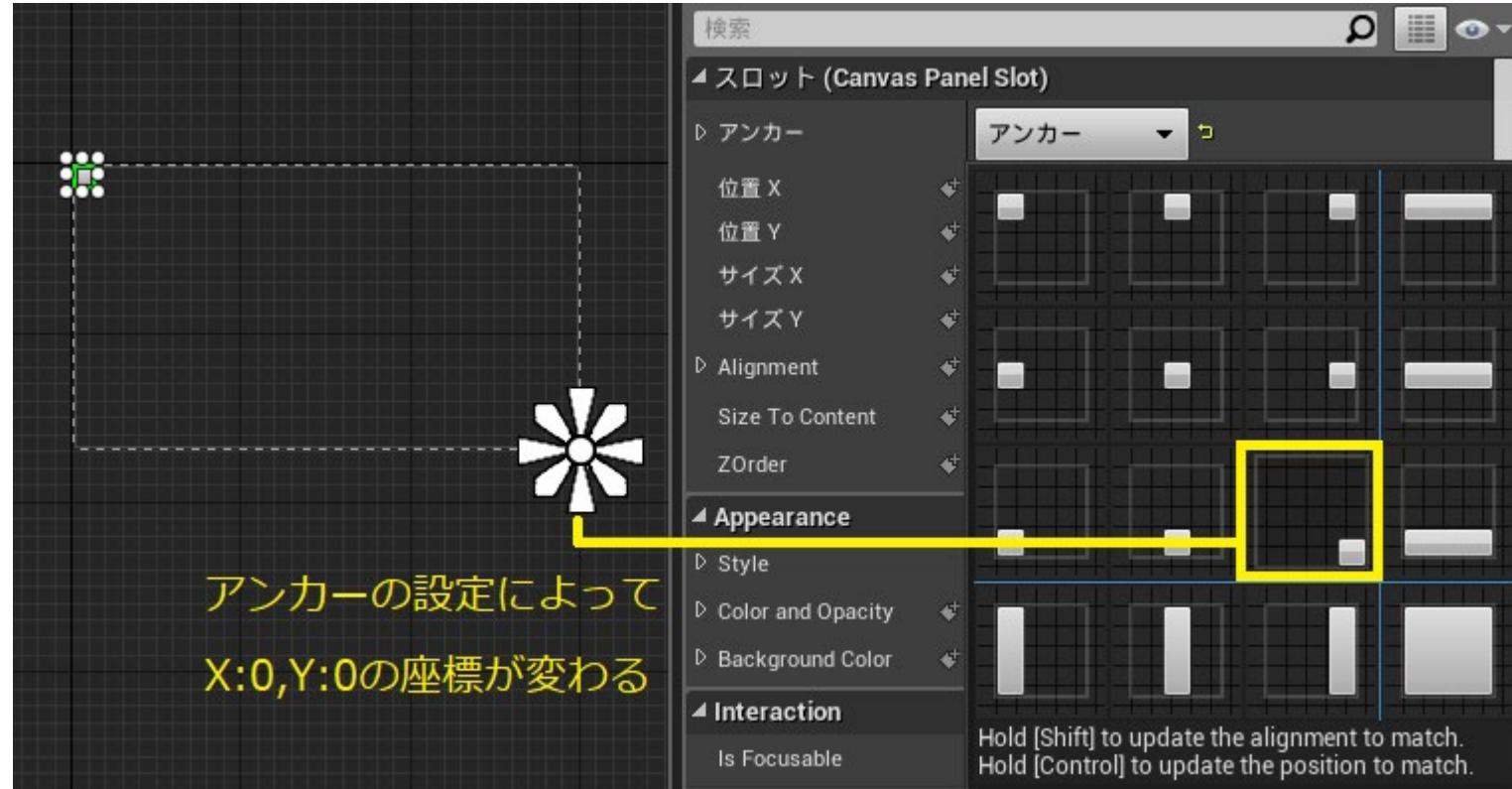


ボタンに配置したテキストも
上下や左右の揃え方も設定することができる

ウィジェットの座表



アンカーに設定によって、X:0,Y:0の位置が変わる

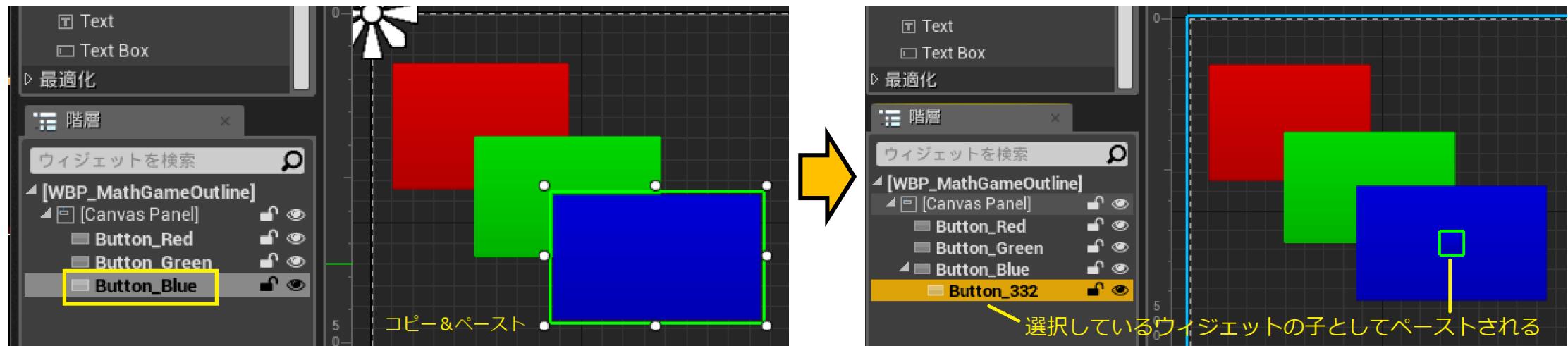


アンカー

<https://docs.unrealengine.com/latest/JPN/Engine/UMG/UserGuide/Anchors/index.html>

ウィジェットのコピー & ペーストの注意

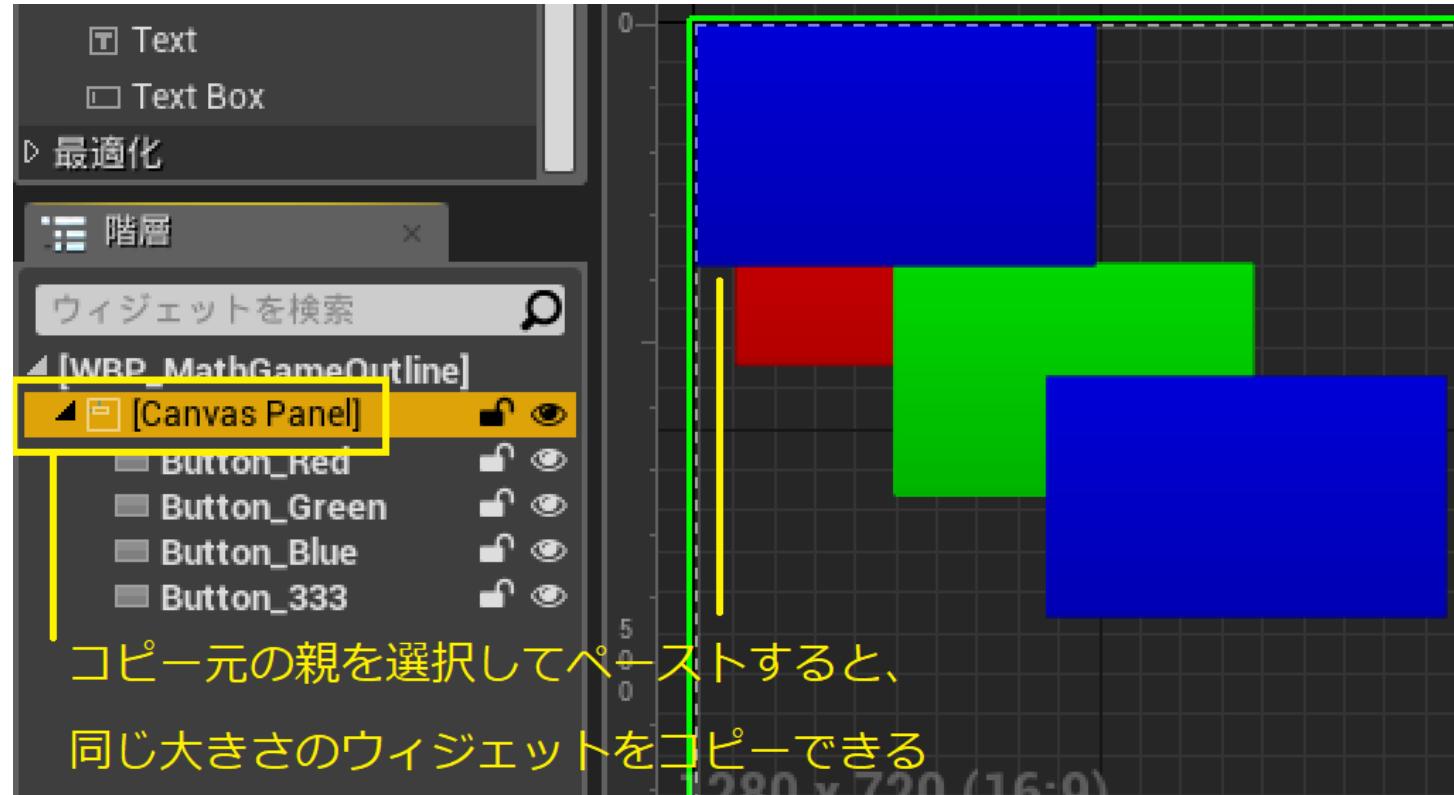
ウィジェットのコピー & ペーストは少し癖があるので注意



ウィジェットを選択してコピー & ペースト

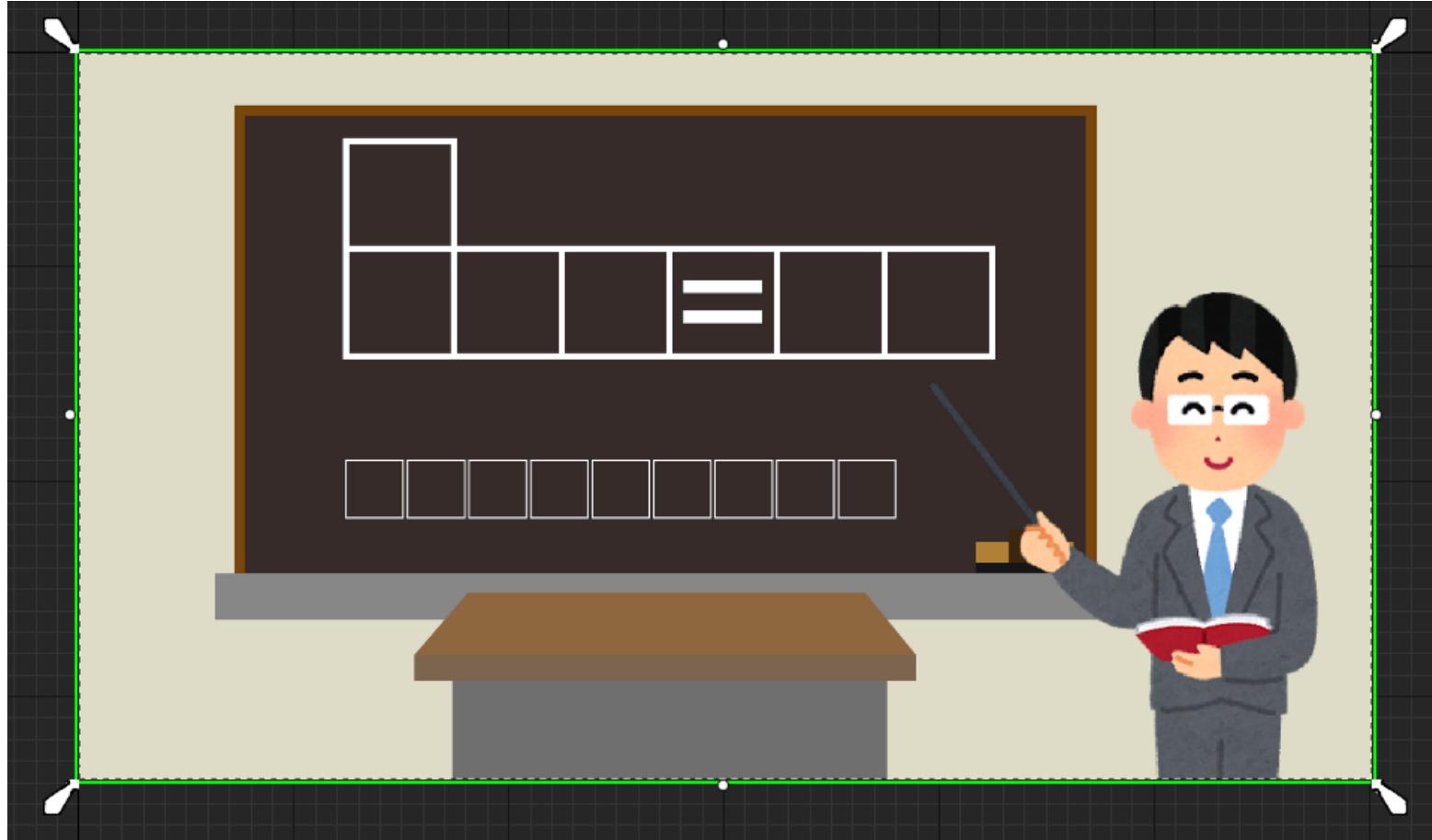
選択しているウィジェットの子として
複製されるので、同じ大きさにならない

同じ大きさでコピー & ペーストする際には
コピー元の親を選択して、ペースト

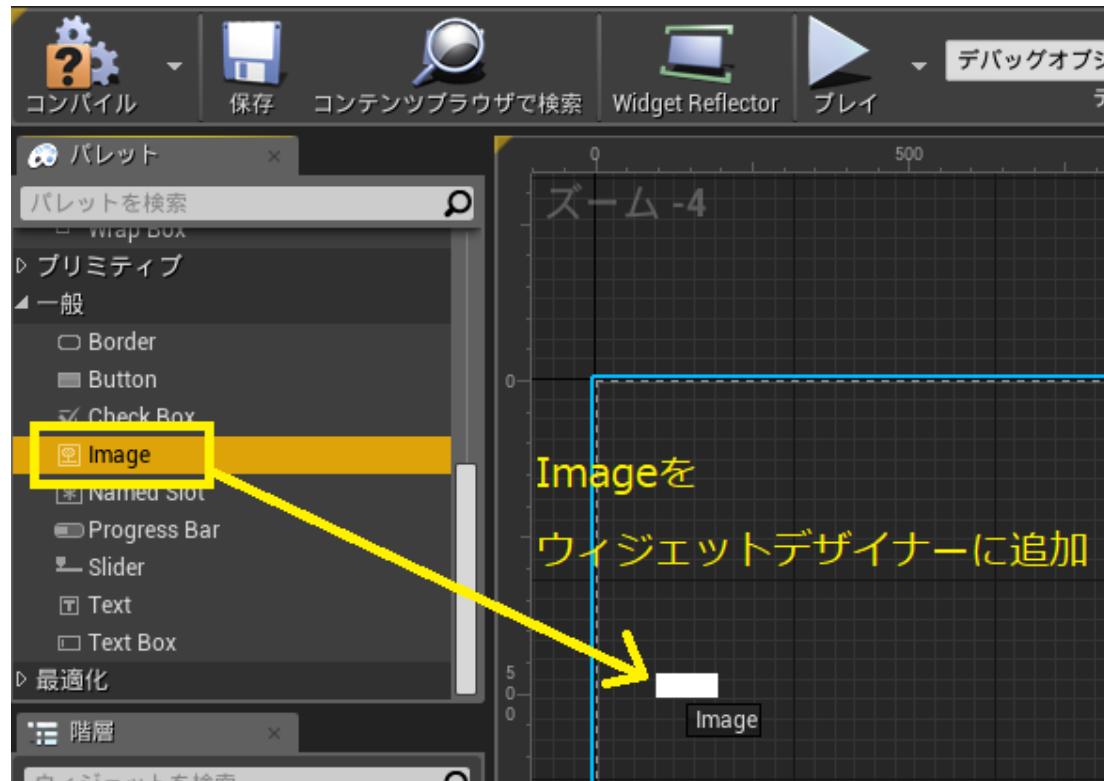


コピー元の親を選択してペースト
ウィジェットが同一階層に複製されるので、同じ大きさになる

Canvas_Panelに隙間なく
画像が表示されるように設定する

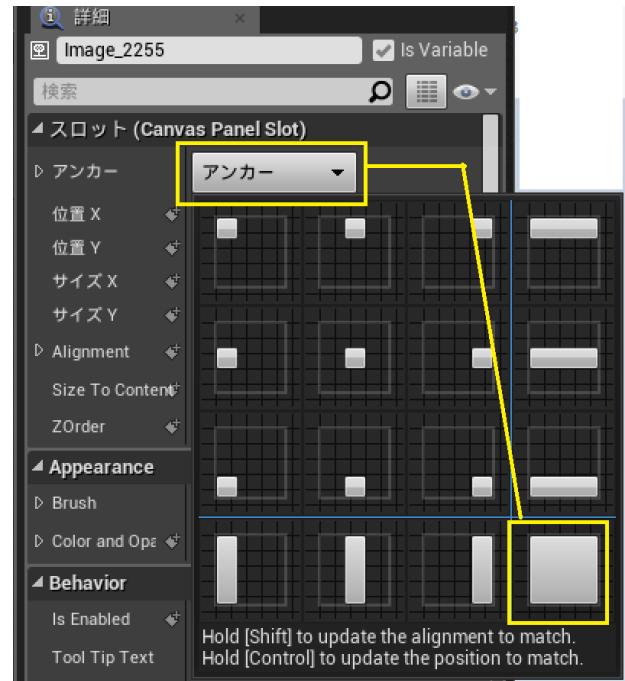


Imageをウィジェットデザイナーに追加

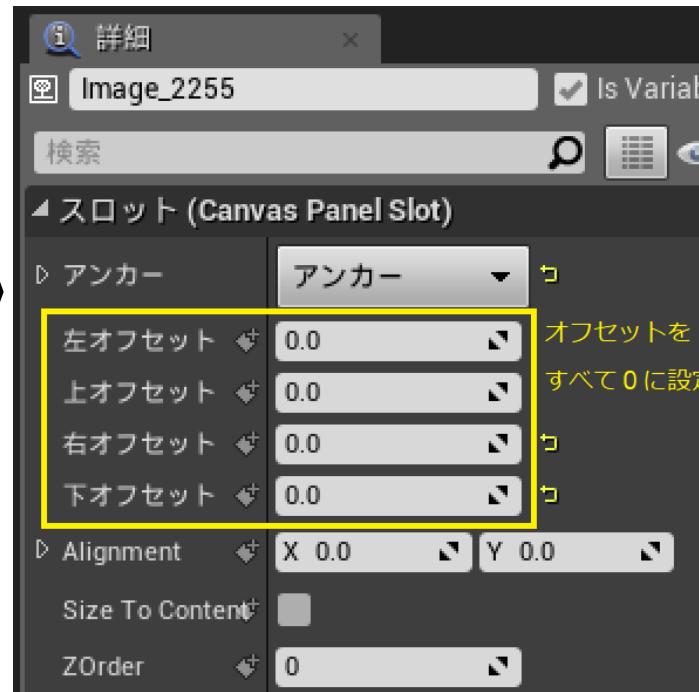


パレットタブ > 一般 > Imageを
ウィジェットデザイナーにドラッグ&ドロップ

Canvas_Panelに隙間なく 画像が表示されるように設定する



アンカーをクリック
> 右下のアイコンをクリック



オフセットをすべて 0 に設定

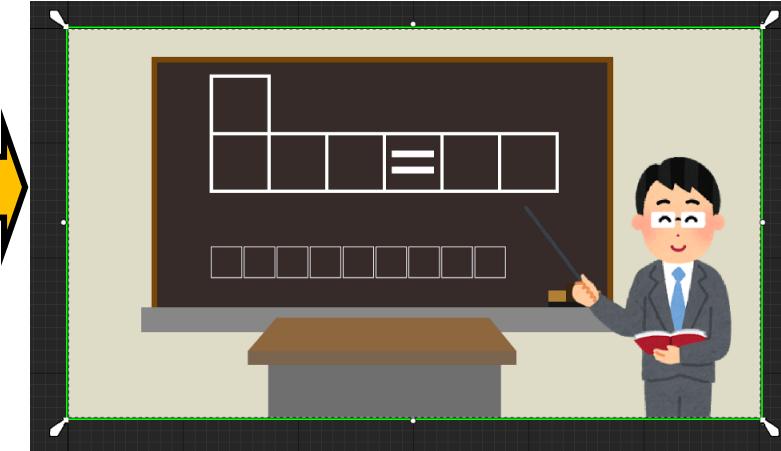


Canvas_Panelに隙間なく拡がる

Imageに画像(テクスチャ)を設定する BrushのImageにテクスチャを設定



Brush > Image > 設定したいテクスチャを設定

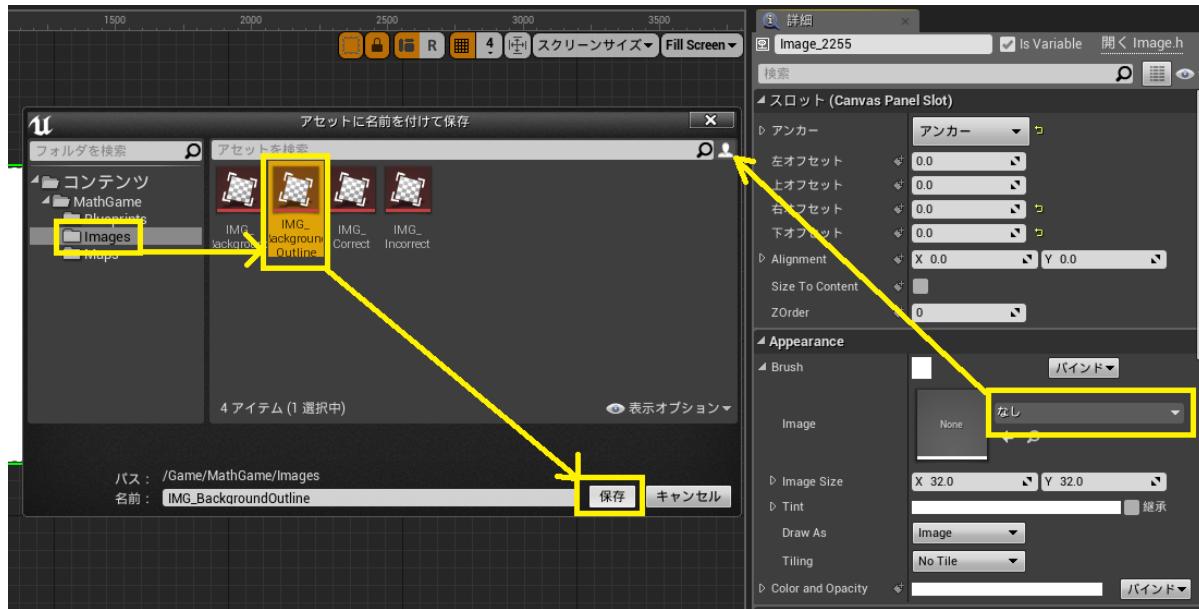


アウトライン画像
が表示される

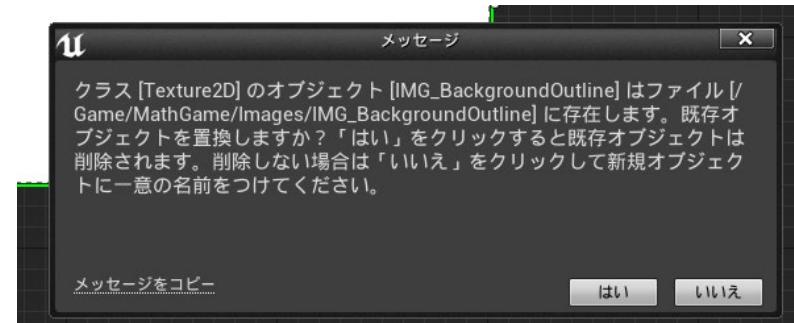


画像を選択してから、Brush > Imageの←をクリック
(リストから画像を選択しない方がいい)

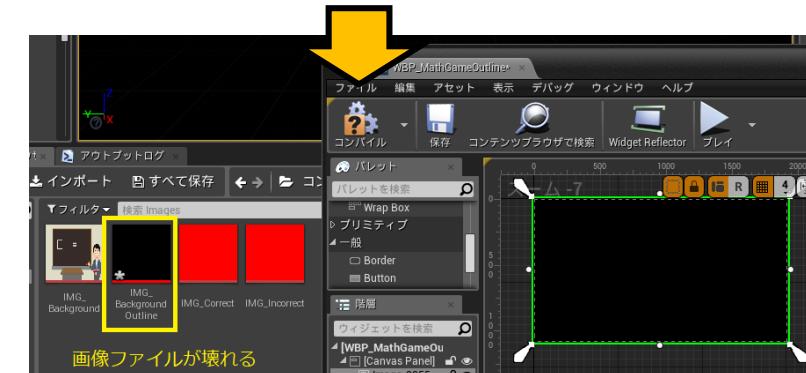
リストを選択した際にファイル選択のダイアログが出たときに、画像ファイルを選択するとファイルが壊れる



Brush > Imageをクリックしたときに
ファイル選択ダイアログが表示される時がある（時々）
ダイアログから画像を選択する



メッセージダイアログで[はい]を選択すると
ファイルが削除される



黒いファイルを削除して、
再度画像をインポートする

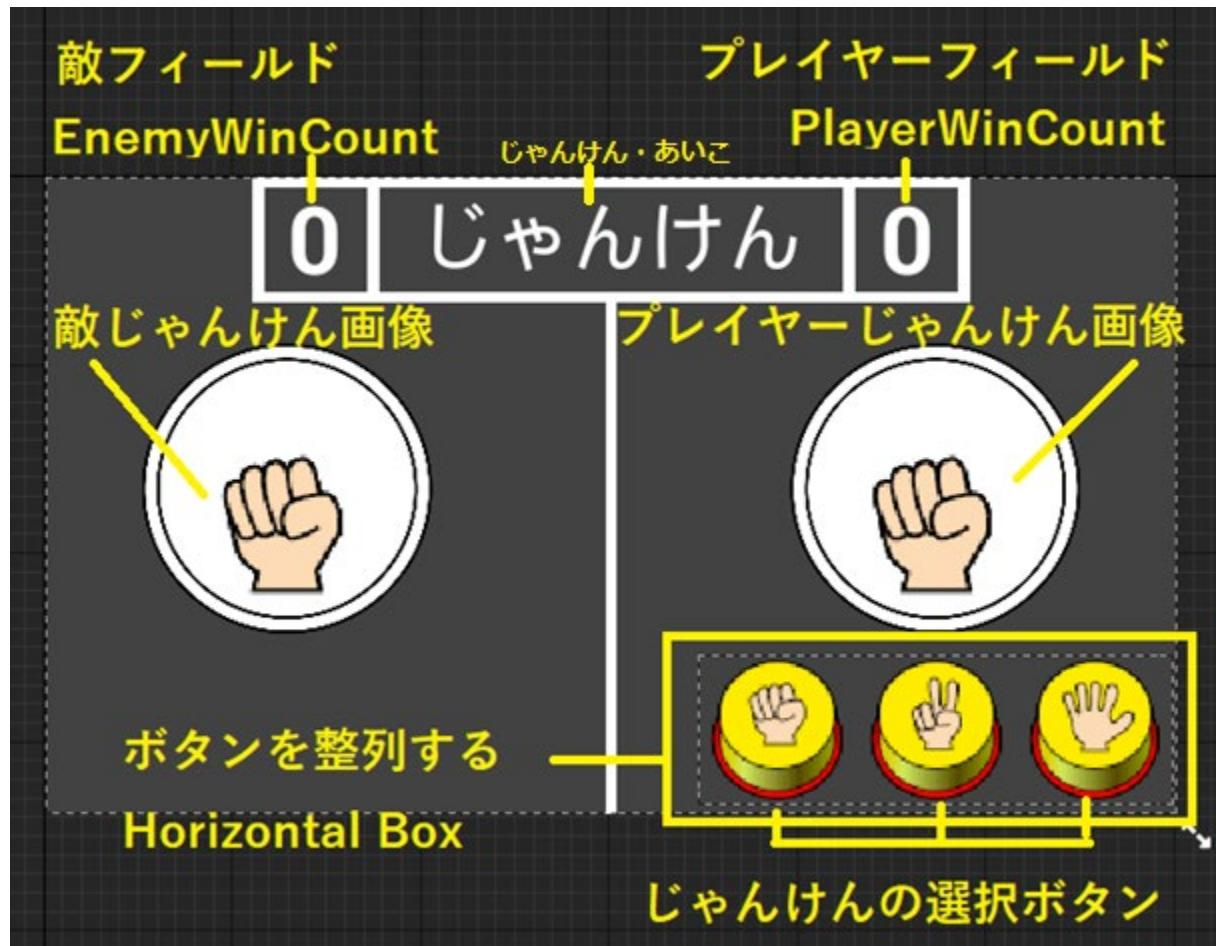
7.じゃんけんゲームのUI作成

7.1 UIに使用する画像のインポート・設定

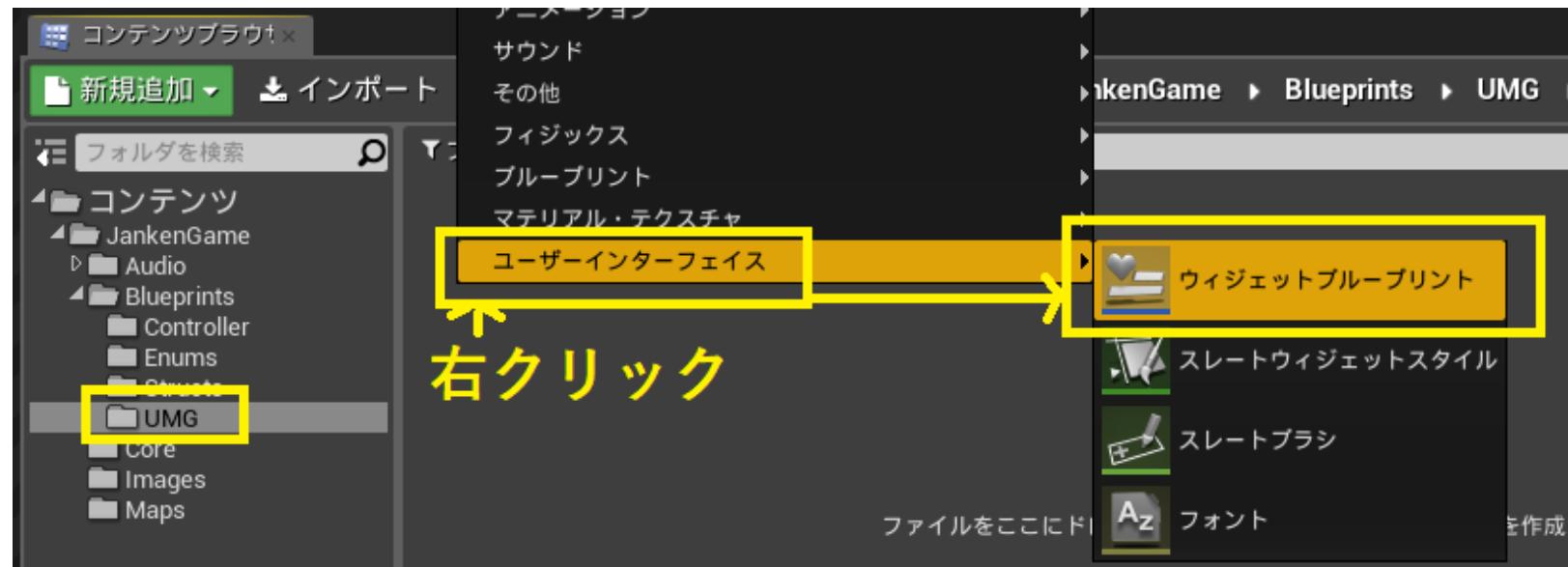
7.2 ウィジェットブループリントの操作のおさらい

7.3 ゲームのじゃんけん選択画面を作成する

じゃんけんの選択画面を作成する



ウィジェットブループリント:WBP_Gameを作成



UMGフォルダを選択

> 右クリック

> ユーザーインターフェース

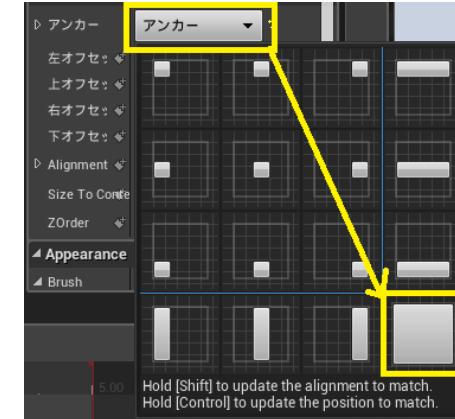
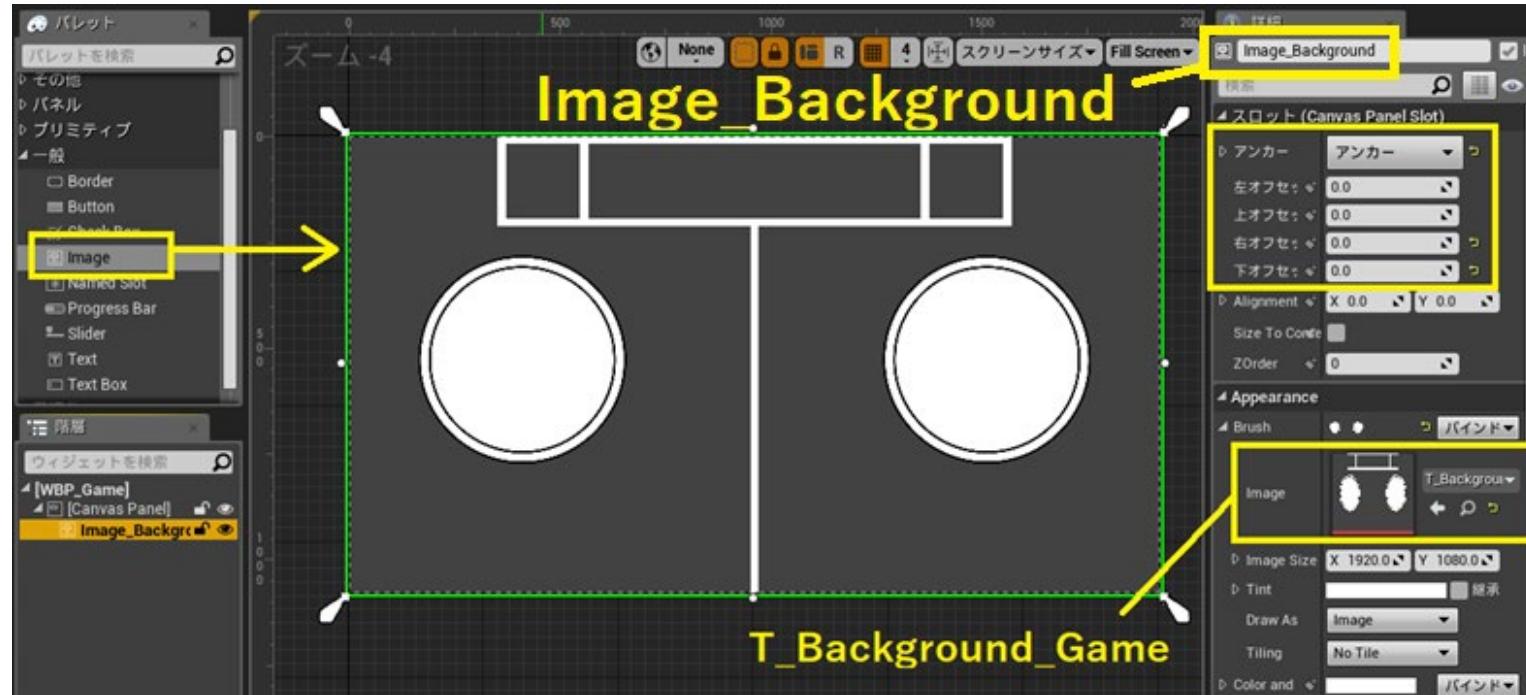
> ウィジェットブループリント



名前をWBP_Gameに設定

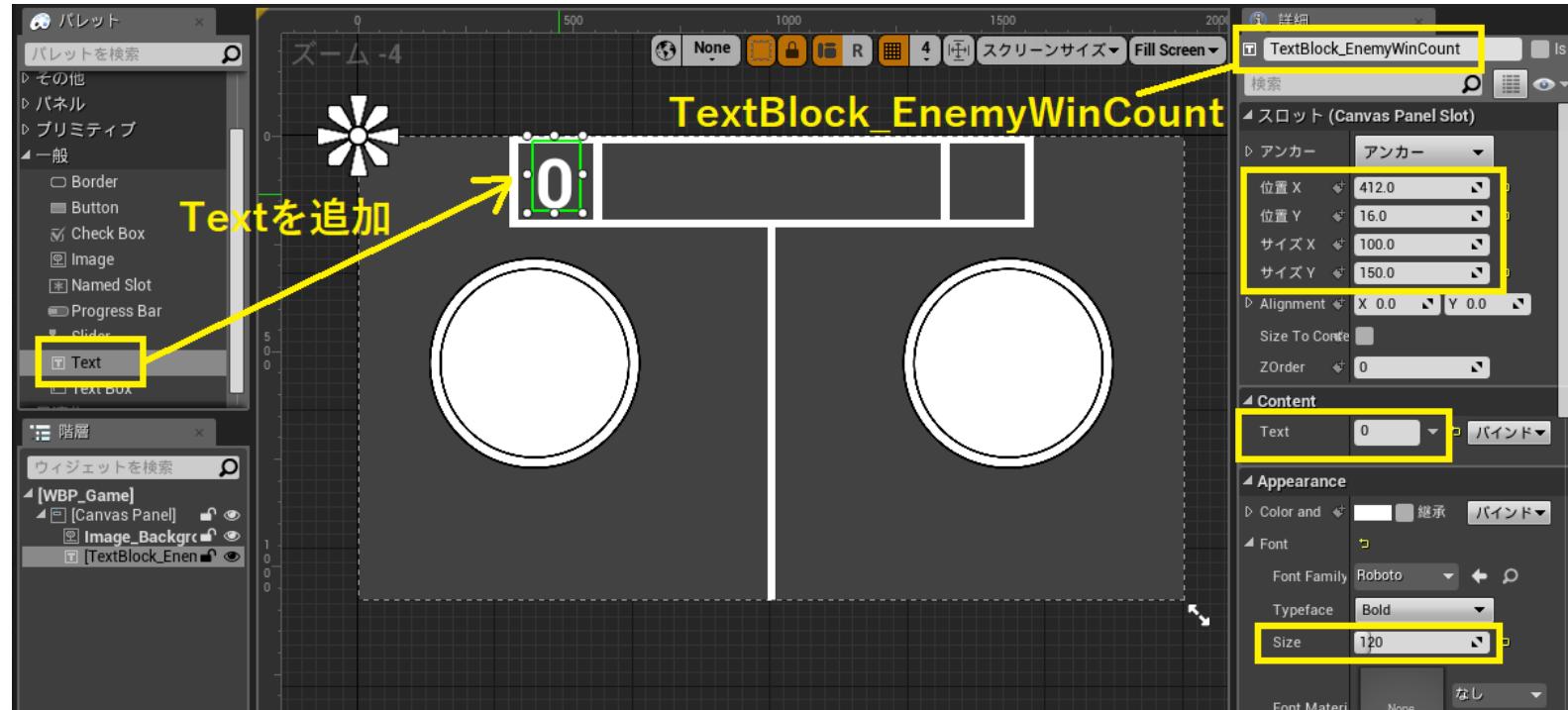
名前をWBP_Gameに設定

ウィジェット:Imageを追加 背景画像を設定する



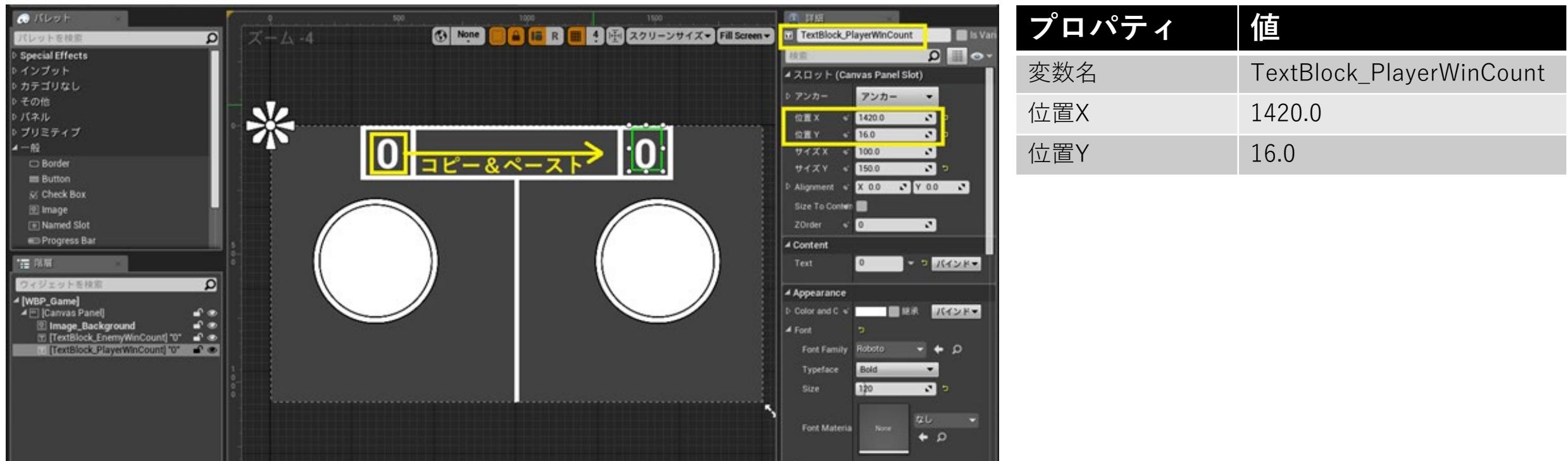
プロパティ	値
変数名	Image_Background
左オフセット	0.0
上オフセット	0.0
右オフセット	0.0
下オフセット	0.0
Brush > Image	T_Background_Game

ウィジェット:Textを追加 敵の勝利カウントを設定する



プロパティ	値
変数名	TextBlock_EnemyWinCount
位置X	412.0
位置Y	16.0
サイズX	100.0
サイズY	150.0
Text	0
Font > Size	120

TextBlock_EnemyWinCountをコピー & ペースト プレイヤーの勝利カウントを設定する

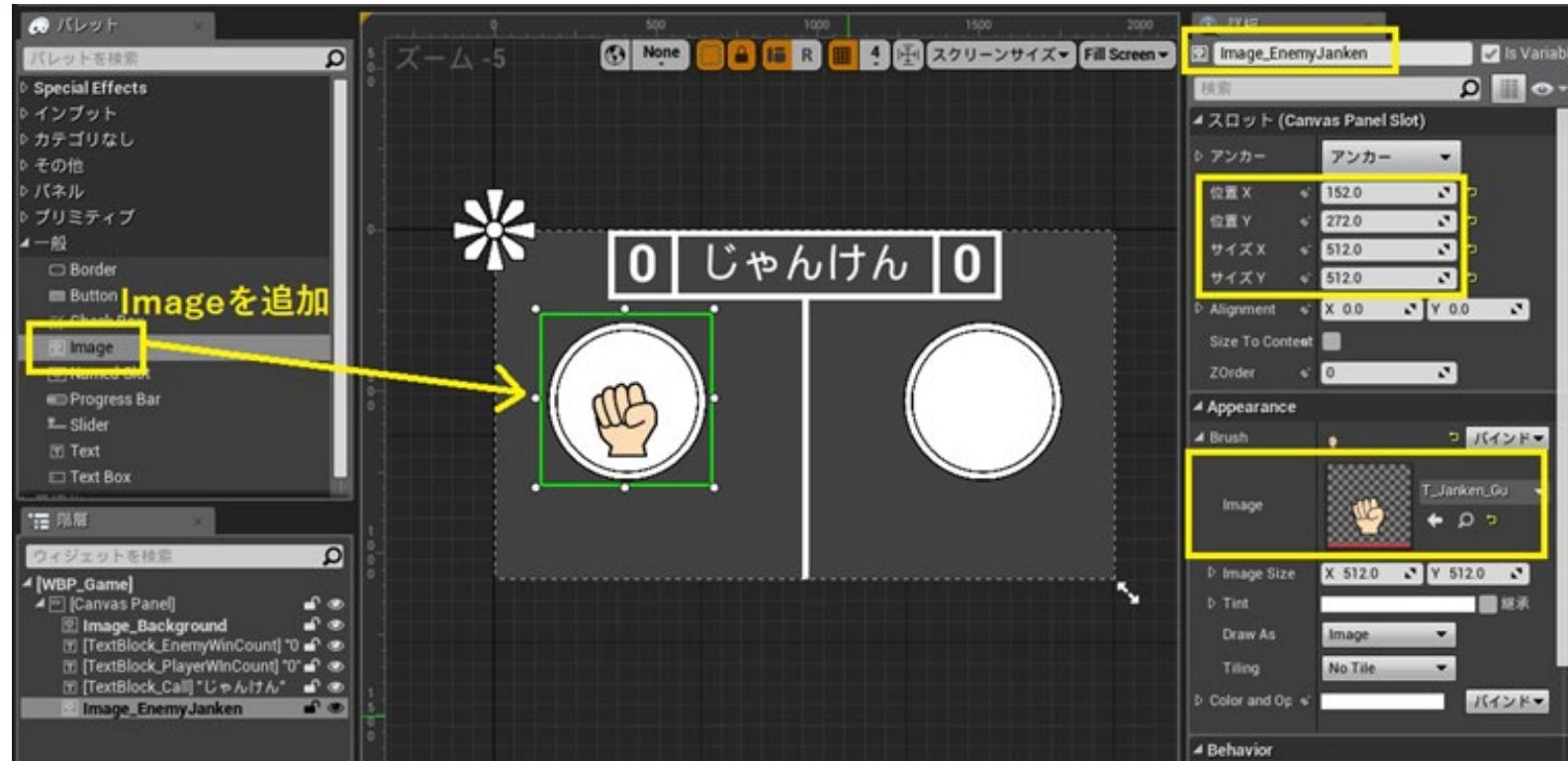


TextBlock_EnemyWinCountをコピー & ペースト
"じゃんけん"や"あいこ"のかけ声を表示する

The screenshot shows the Unreal Engine 4 Editor interface. On the left is the Content Browser with a folder named [WBP_Game] containing [Canvas Panel], [Image_Background], [TextBlock_EnemyWinCount] "0", [TextBlock_PlayerWinCount] "0", and [TextBlock_Call] "じゃんけん". The main area is the Canvas Panel where a text block labeled "じゃんけん" is centered between two circular icons. A yellow arrow points from the text block to the text "コピー & ペースト" (Copy & Paste) located below it. To the right is the Properties panel for the selected TextBlock_Call component. The properties listed are:

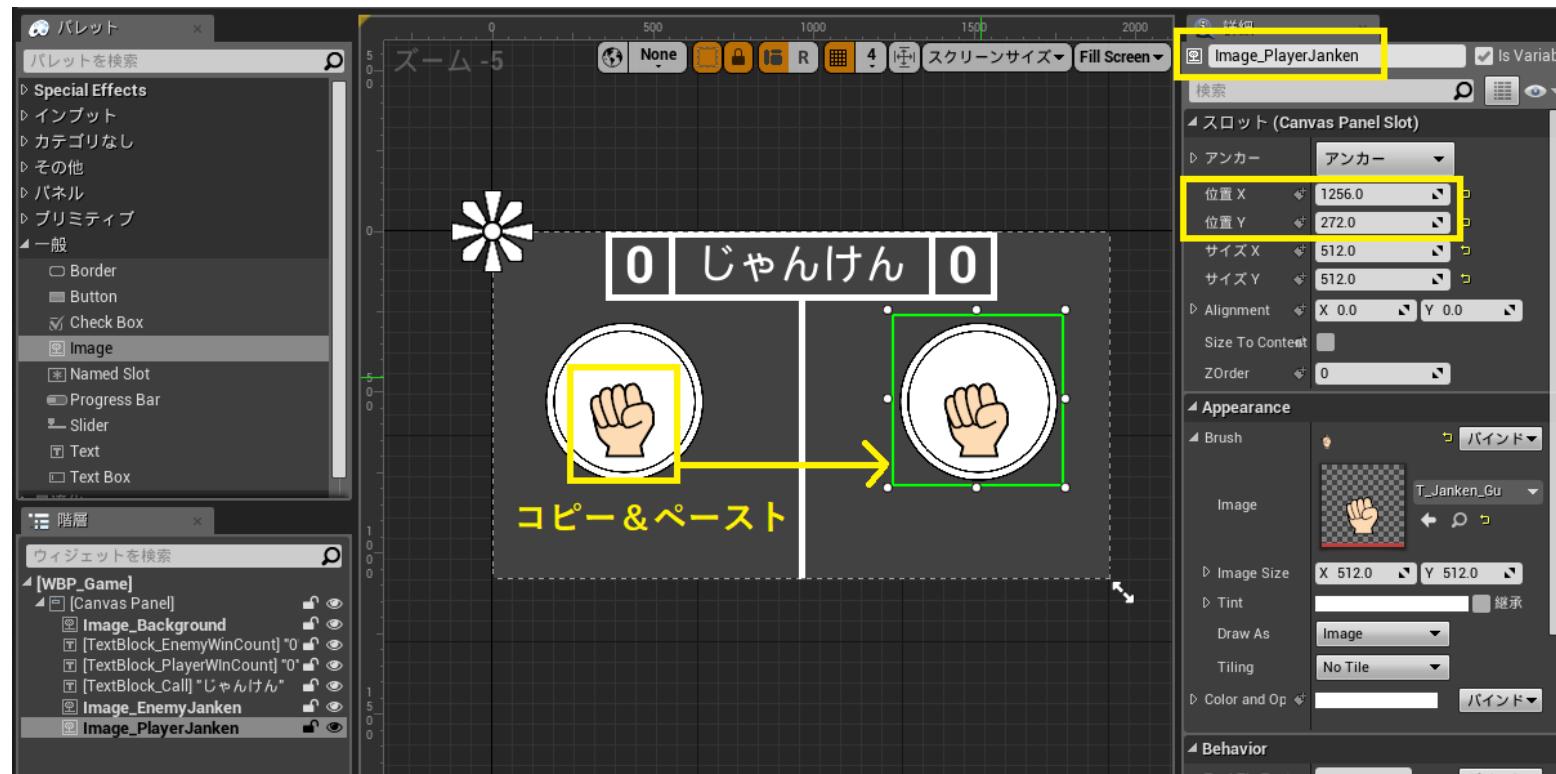
プロパティ	値
変数名	TextBlock_Call
位置X	572.0
位置Y	28.0
サイズX	775.0
Font > Size	100
Text	じゃんけん
Justification	テキスト中央揃え

ウィジェット:Imageを追加 敵のじゃんけん画像を設定する



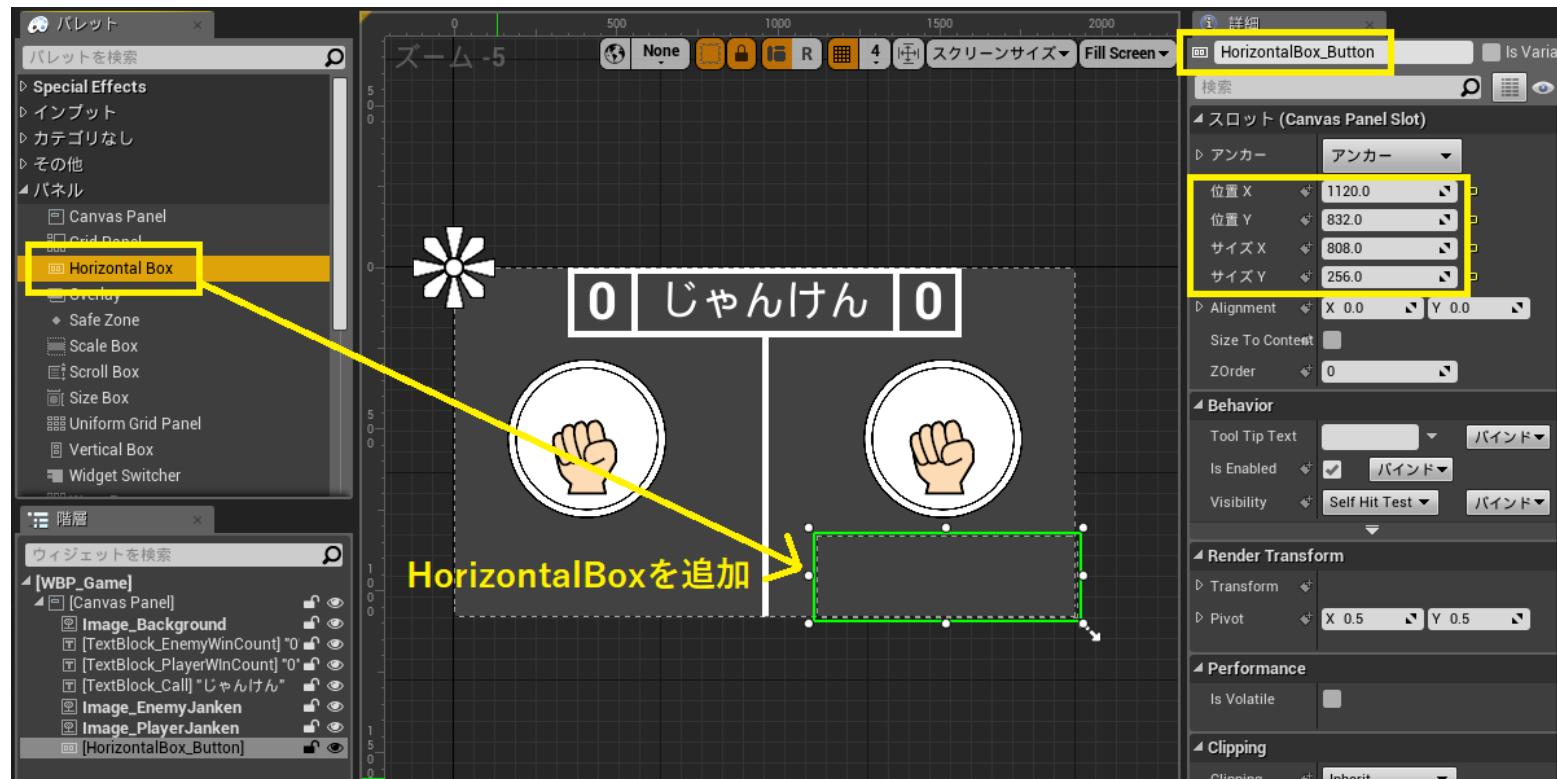
プロパティ	値
変数名	Image_EnemyJanken
位置X	152.0
位置Y	272.0
サイズX	512.0
サイズY	512.0
Brush > Image	T_Janken_Gu

Image_EnemyJankenをコピー & ペースト プレイヤーのじゃんけん画像を設定する



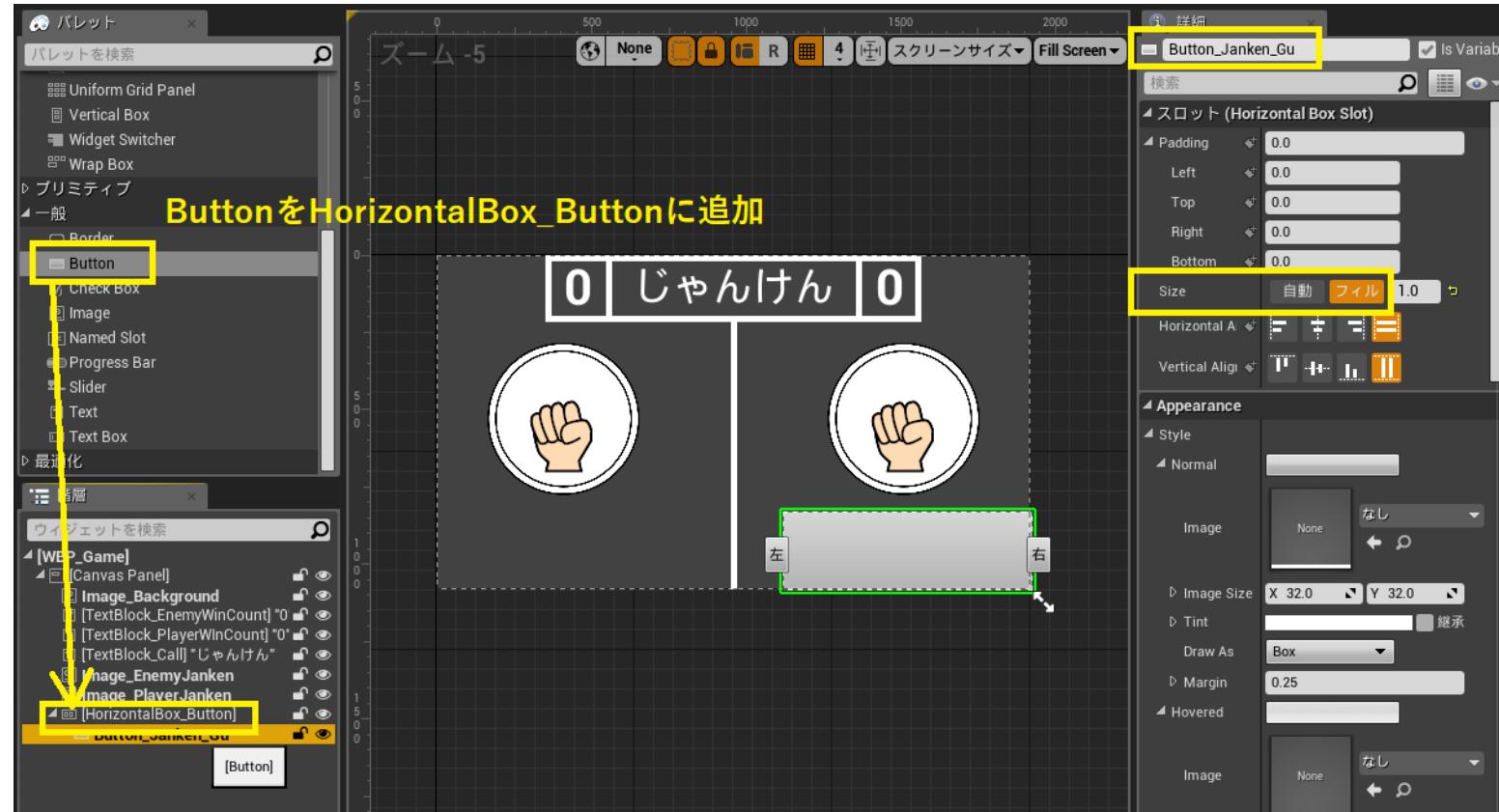
プロパティ	値
変数名	Image_PlayerJanken
位置X	1256.0
位置Y	272.0

ウィジェット：HorizontalBox_Buttonを追加



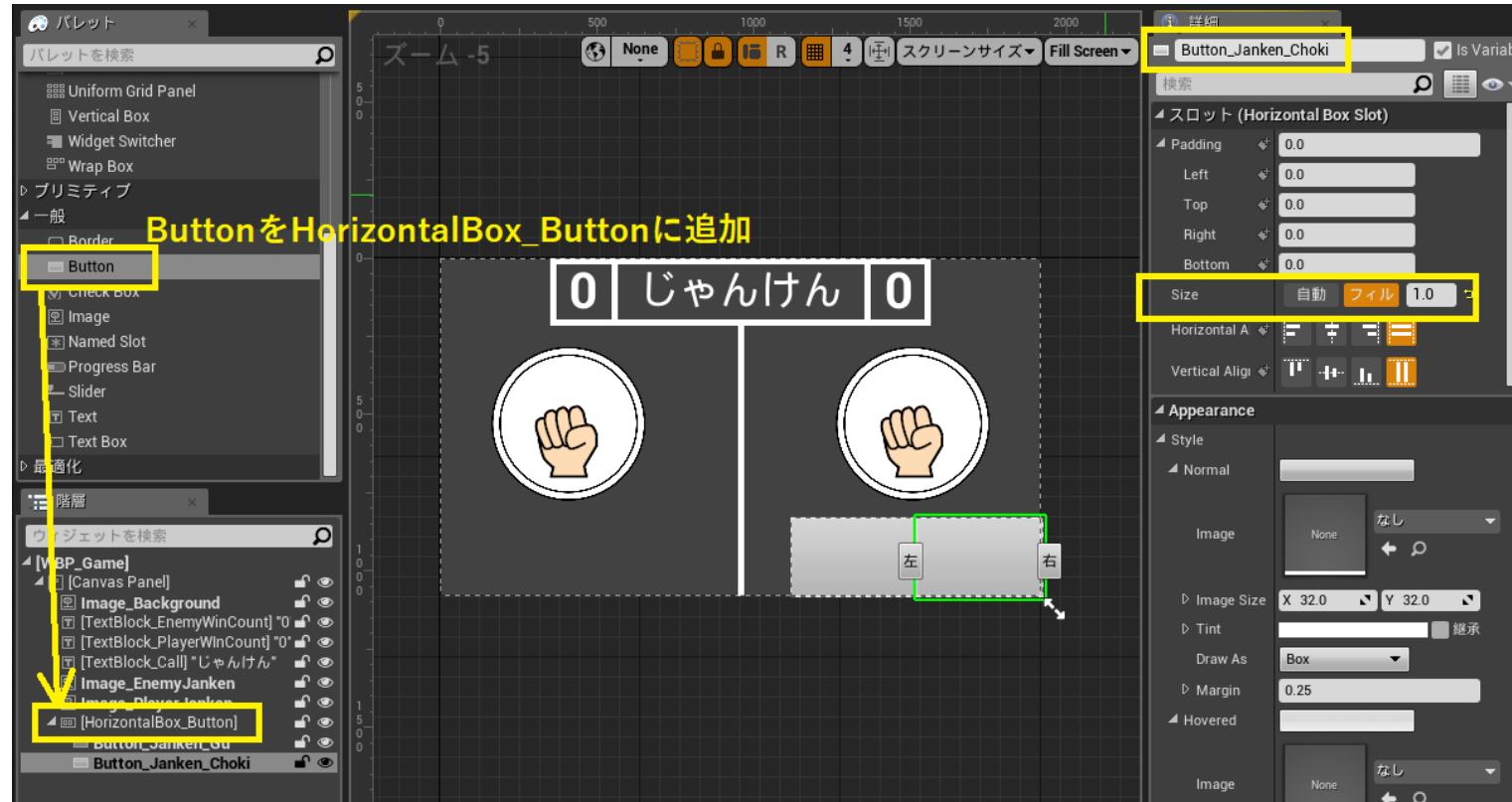
プロパティ	値
変数名	HorizontalBox_Button
位置X	1110.0
位置Y	810.0
サイズX	808
サイズY	256

ウィジェット:ButtonをHorizontalBox_Buttonに追加 Button_Janken_Guの設定



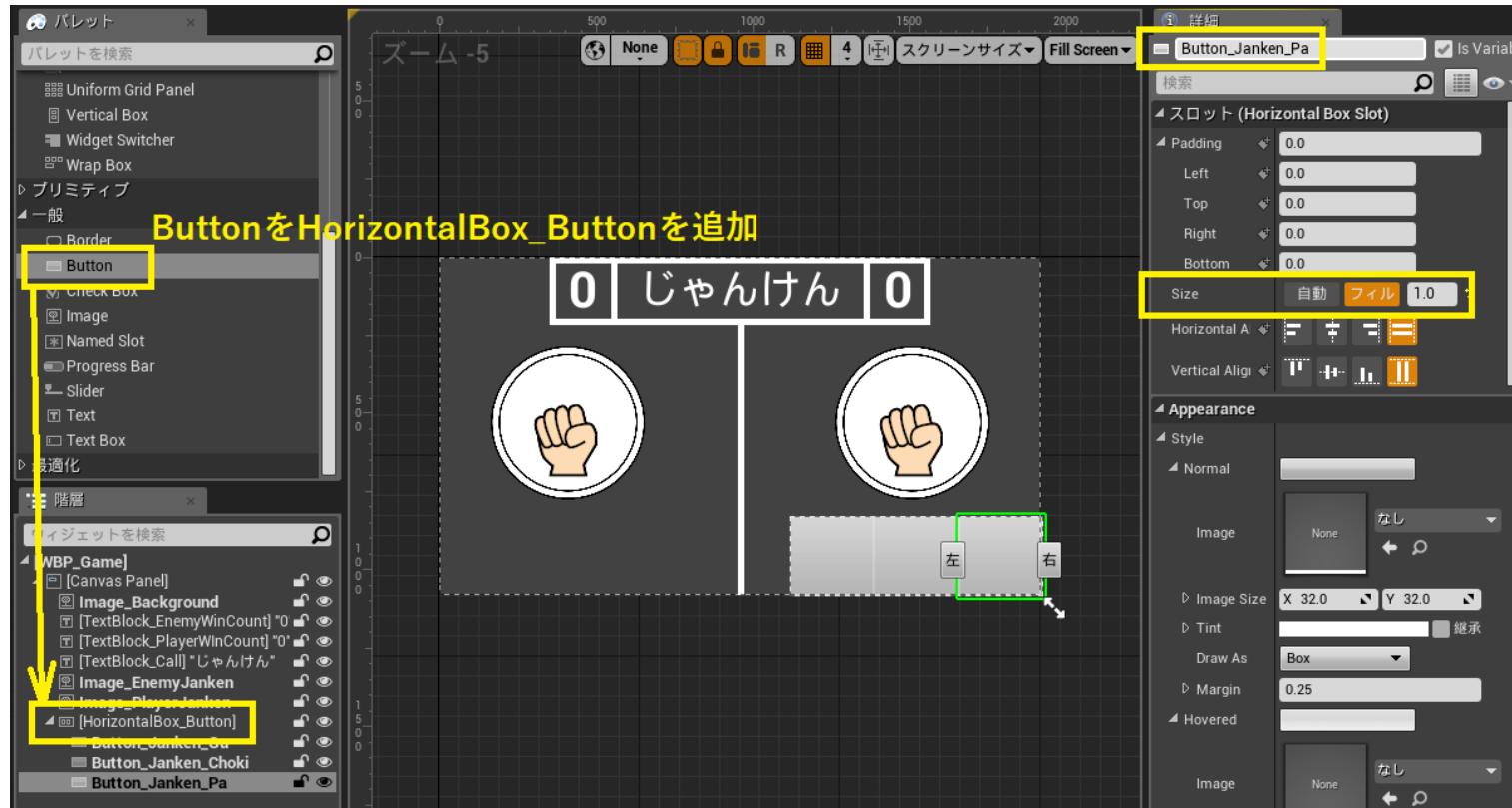
プロパティ	値
変数名	Button_Janken_Gu
サイズ	フィル

ウィジェット:ButtonをHorizontalBox_Buttonに追加 Button_Janken_Chokiの設定



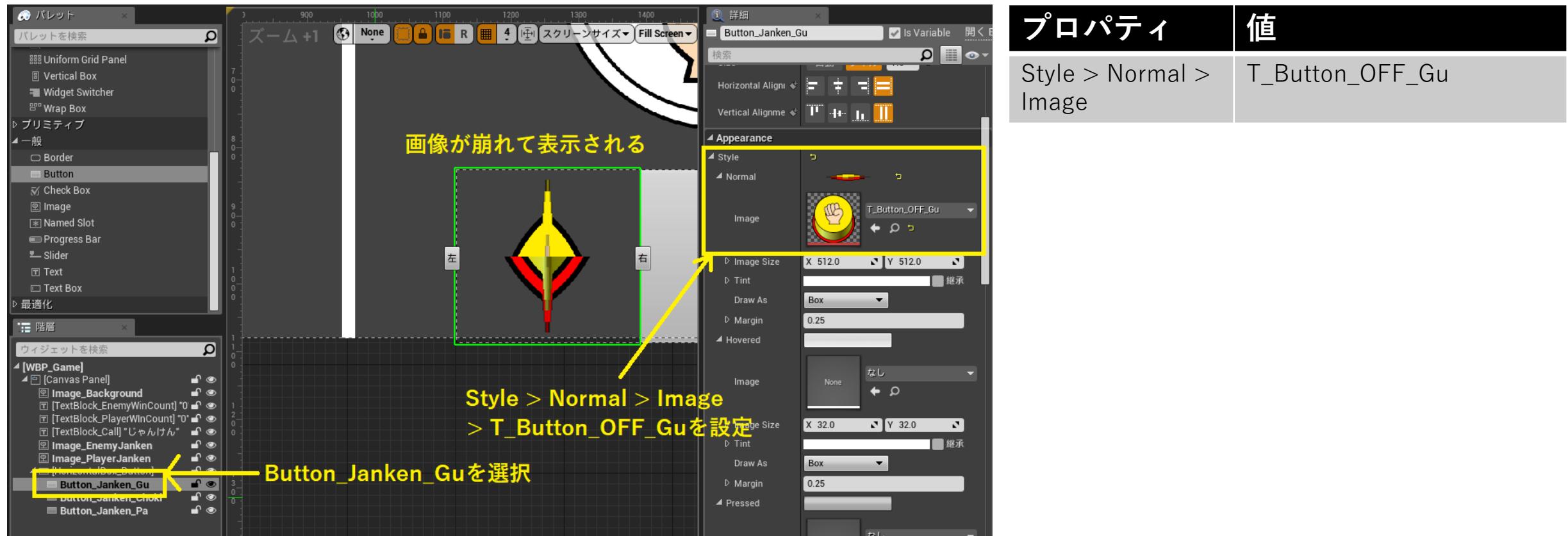
プロパティ	値
変数名	Button_Janken_Choki
サイズ	フィル

ウィジェット:ButtonをHorizontalBox_Buttonに追加 Button_Janken_Paの設定

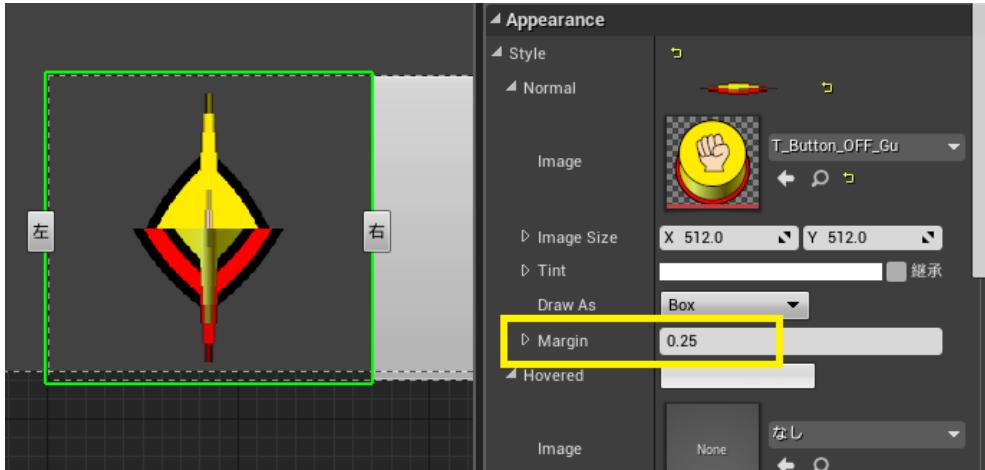


プロパティ	値
変数名	Button_Janken_Pa
サイズ	フィル

ボタンの画像を設定する Button_Janken_GuのStyle Normalに画像を設定



Marginを0.0に設定

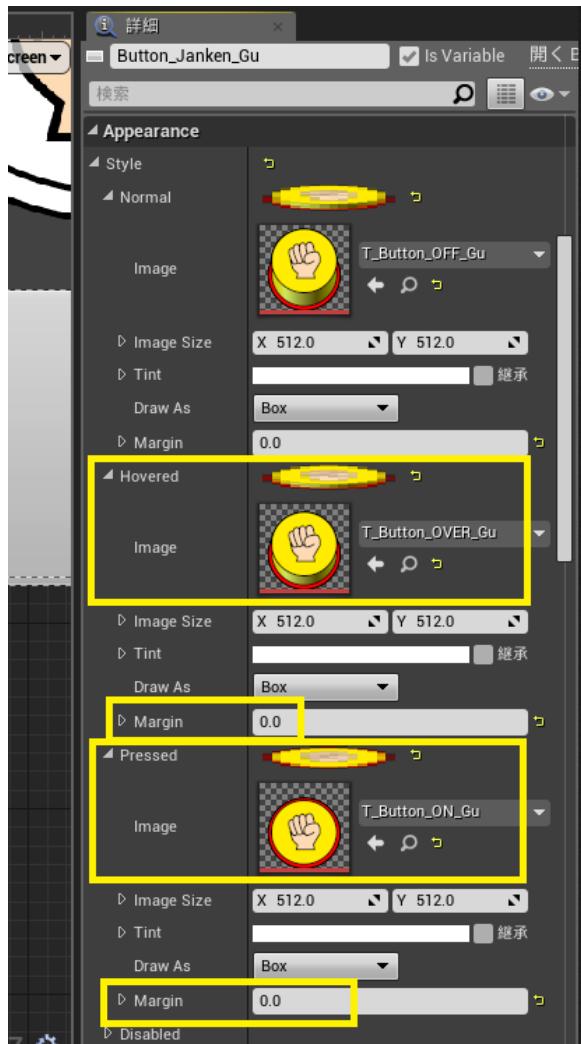


プロパティ	値
Style > Normal > Margin	0.0

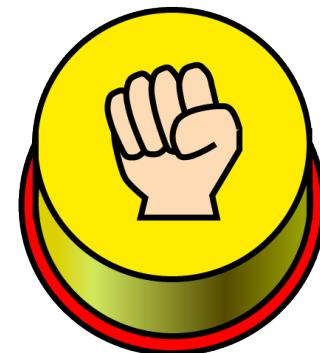


Marginを0.0に設定すると正しく表示される

HoverdとPressedに画像を設定



プロパティ	値
Style > Hovered > Image	T_Button_OVER_Gu
Style > Hovered > Margin	0.0
Style > Pressed > Image	T_Button_ON_Gu
Style > Pressed > Margin	0.0



Style > Normal > Image
T_Button_OFF_Gu

何もない時の画像表示



Style > Hovered > Image
T_Button_OVER_Gu

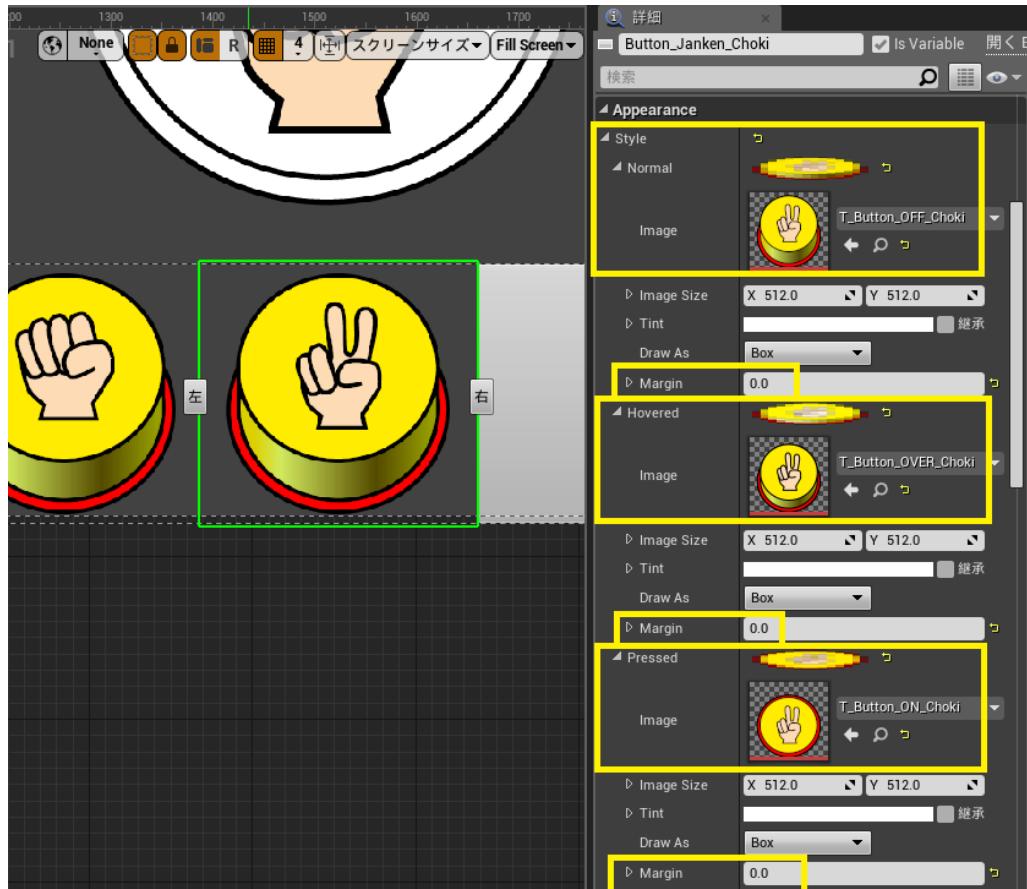
マウスが画像に重なった時



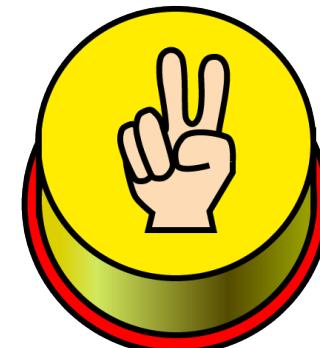
Style > Pressed > Image
T_Button_ON_Gu

画像をマウスがクリックした時

Button_Janken_Chokiに画像を設定

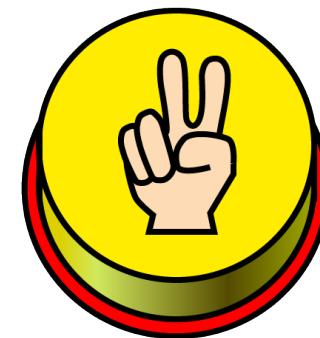


プロパティ	値
Style > Normal > Image	T_Button_OFF_Choki
Style > Normal > Margin	0.0
Style > Hovered > Image	T_Button_OVER_Choki
Style > Hovered > Margin	0.0
Style > Pressed > Image	T_Button_ON_Choki
Style > Pressed > Margin	0.0



Style > Normal > Image
T_Button_OFF_Choki

何もない時の画像表示



Style > Hovered > Image
T_Button_OVER_Choki

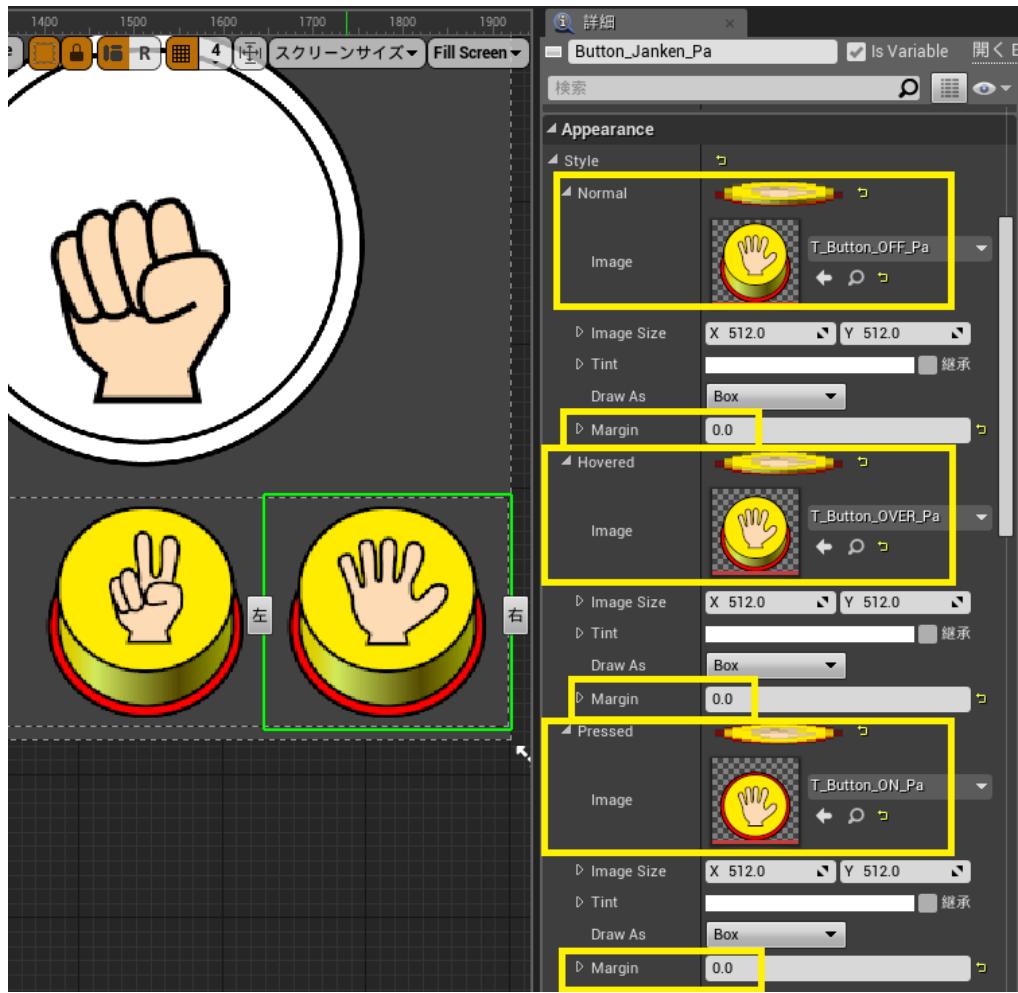
マウスが画像に重なった時



Style > Pressed > Image
T_Button_ON_Choki

画像をマウスがクリックした時

Button_Janken_Paに画像を設定



プロパティ	値
Style > Normal > Image	T_Button_OFF_Pa
Style > Normal > Margin	0.0
Style > Hovered > Image	T_Button_OVER_Pa
Style > Hovered > Margin	0.0
Style > Pressed > Image	T_Button_ON_Pa
Style > Pressed > Margin	0.0



Style > Normal > Image
T_Button_OFF_Choki

何もない時の画像表示



Style > Hovered > Image
T_Button_OVER_Choki

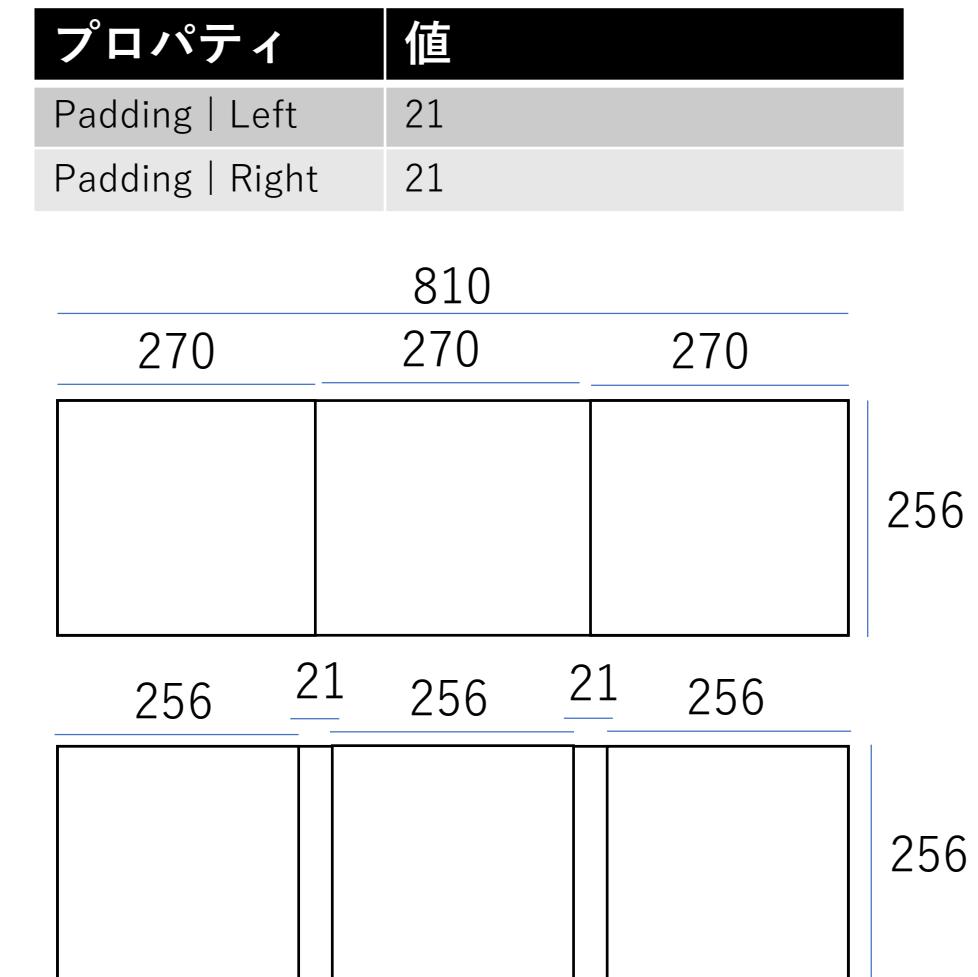
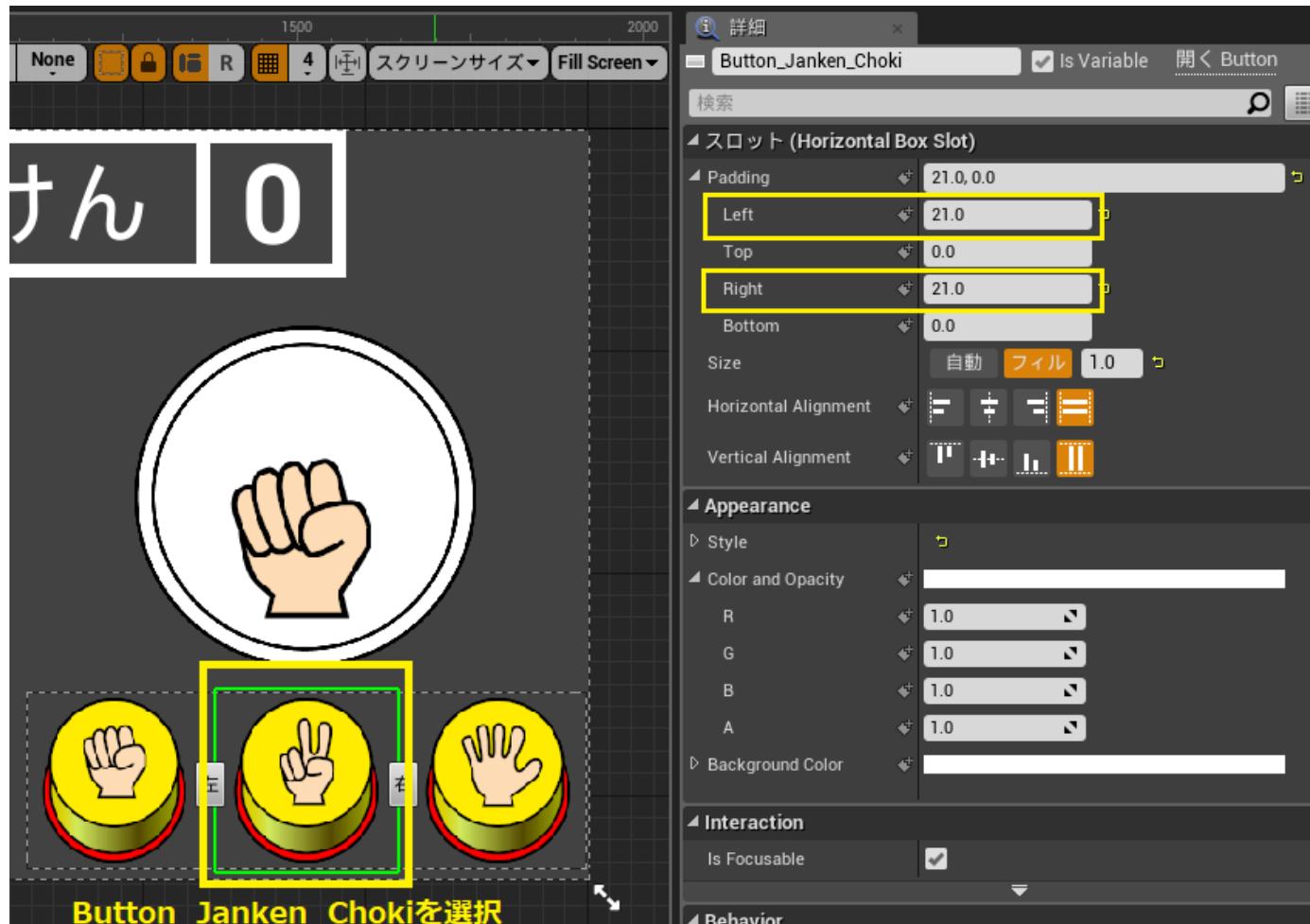
マウスが画像に重なった時



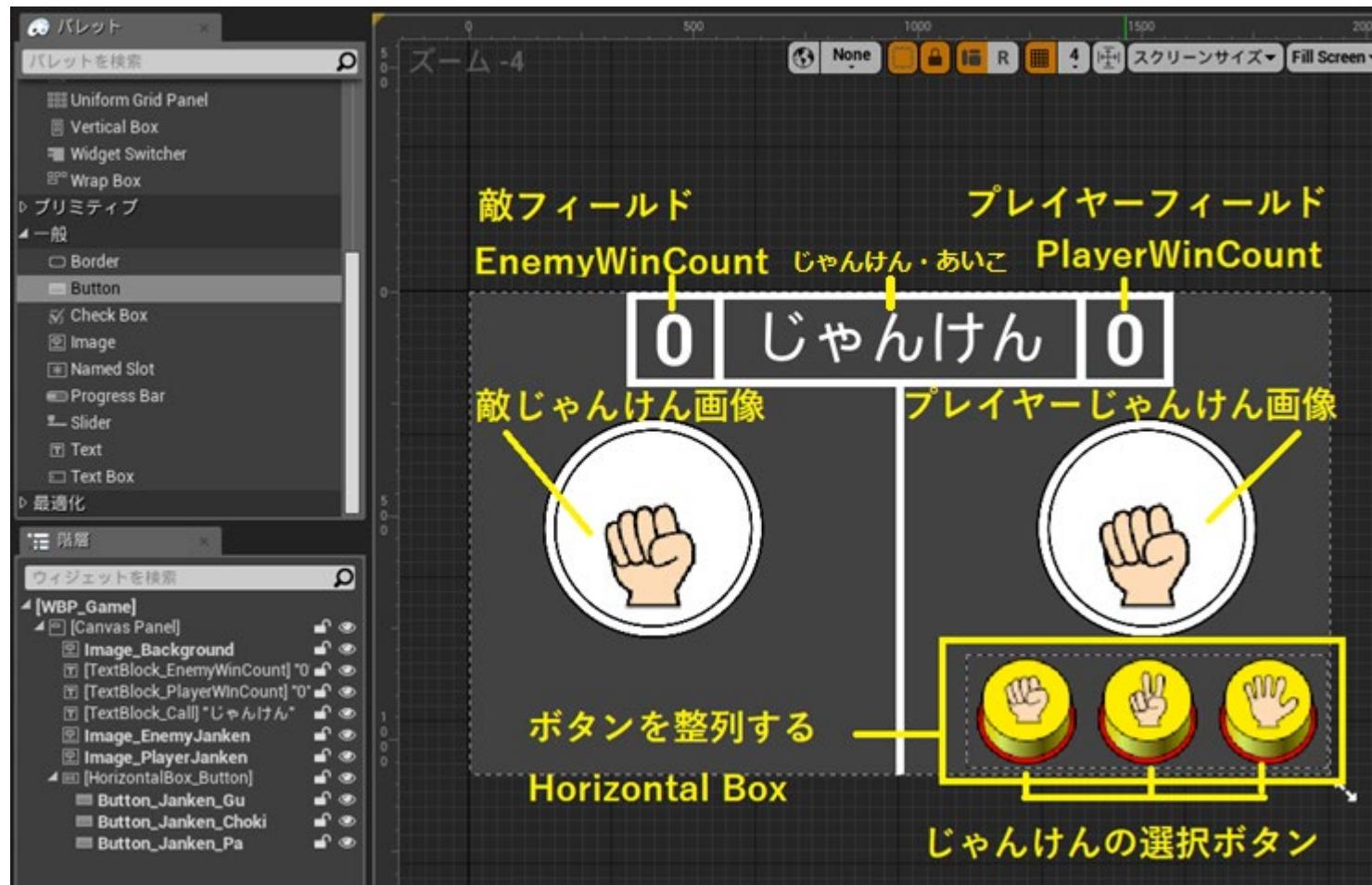
Style > Pressed > Image
T_Button_ON_Choki

画像をマウスがクリックした時

Button_Janken_ChokiのPaddingを設定 ボタンのサイズが正方形になるように調整する



じゃんけんの選択画面完成



8. GameMode, GameStateを作成して、
ゲームの状態を中継する

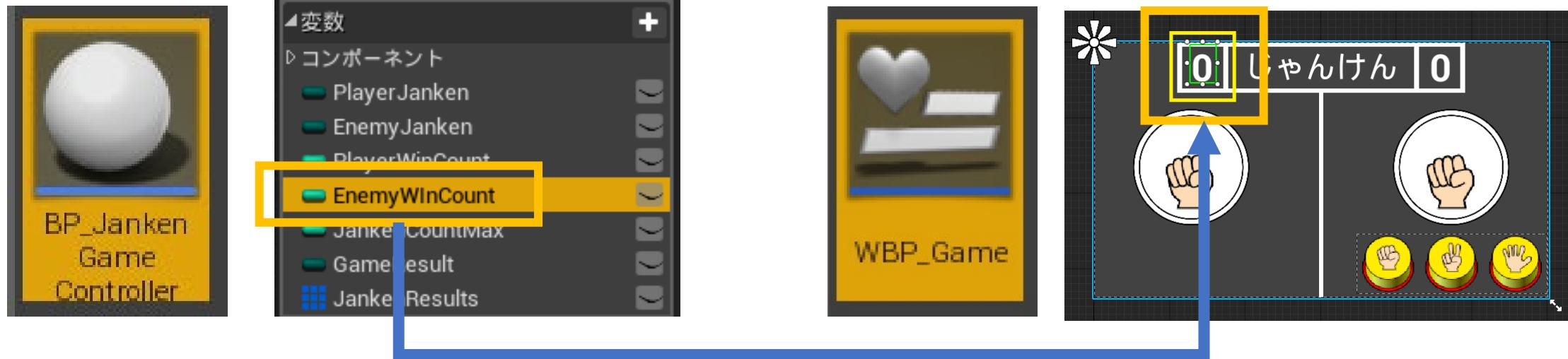
8. GameMode, GameStateを作成して、ゲームの状態を中継する

- 8.1 ウィジェットブループリントとゲームロジックの状態を中継する仕組みを作成する
- 8.2 BP_GameStateを作成する
- 8.3 BP_GameModeを作成する
- 8.4 WBP_Gameのウィジェットにバインドを作成して、BP_GameStateの値を反映させる
- 8.5 WBP_Gameを表示する
- 8.6 ゲームの状態を中継する仕組みの説明
- 8.7 BP_JankenGameControllerの変数をBP_GameStateに移行する
- 8.8 BP_GameStateに関数Initializeを作成し、初期化処理を置き換える
- 8.9 BP_JankenGameControllerの処理をBP_GameStateの変数を使用するように変更する
- 8.10 BP_GameStateのCallJankenStrに文字を設定する
- 8.11 BP_GameStateの変数をWBP_Gameのウィジェットにバインドする

8. GameMode, GameStateを作成して、ゲームの状態を中継する

- 8.1 ウィジェットブループリントとゲームロジックの状態を中継する仕組みを作成する
- 8.2 BP_GameStateを作成する
- 8.3 BP_GameModeを作成する
- 8.4 WBP_Gameのウィジェットにバインドを作成して、BP_GameStateの値を反映させる
- 8.5 WBP_Gameを表示する
- 8.6 ゲームの状態を中継する仕組みの説明
- 8.7 BP_JankenGameControllerの変数をBP_GameStateに移行する
- 8.8 BP_GameStateに関数Initializeを作成し、初期化処理を置き換える
- 8.9 BP_JankenGameControllerの処理をBP_GameStateの変数を使用するように変更する
- 8.10 BP_GameStateのCallJankenStrに文字を設定する
- 8.11 BP_GameStateの変数をWBP_Gameのウィジェットにバインドする

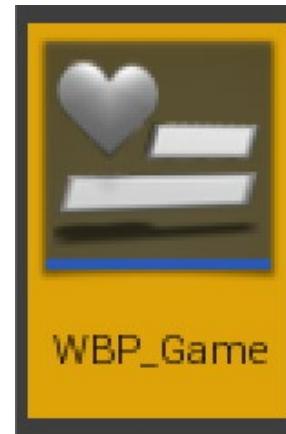
ロジックを書いたブループリントの変数の値を、
ウィジェットブループリントに反映させたい



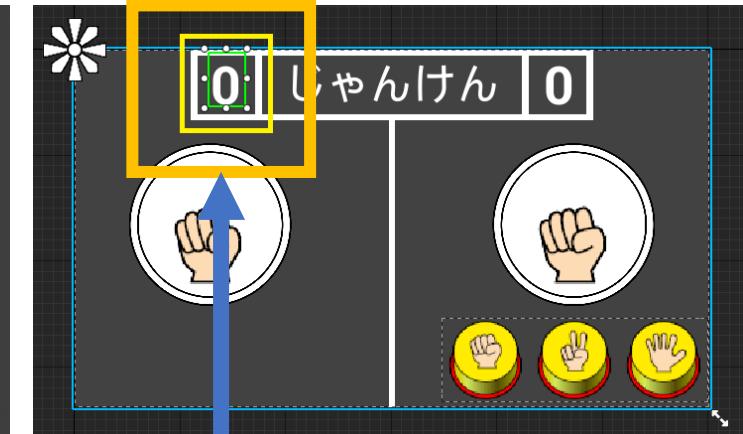
ゲームの状態を保持するGameStateブループリントを作成し、ゲームのロジックが持っていた変数を移植する



GameStateの変数を更新する



GameStateから値を取得する



画面に表示する値は
GameStateが保持する

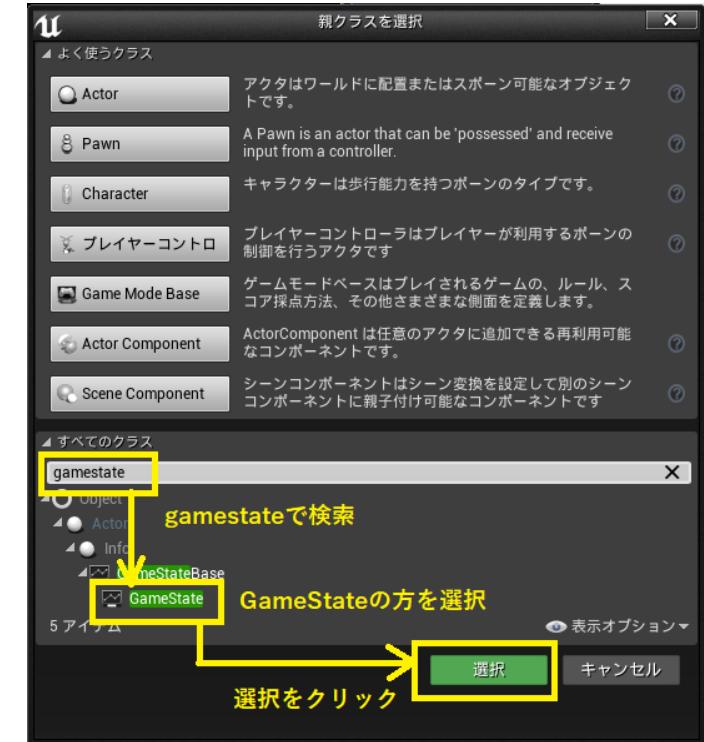
8. GameMode, GameStateを作成して、ゲームの状態を中継する

- 8.1 ウィジェットブループリントとゲームロジックの状態を中継する仕組みを作成する
- 8.2 BP_GameStateを作成する
- 8.3 BP_GameModeを作成する
- 8.4 WBP_Gameのウィジェットにバインドを作成して、BP_GameStateの値を反映させる
- 8.5 WBP_Gameを表示する
- 8.6 ゲームの状態を中継する仕組みの説明
- 8.7 BP_JankenGameControllerの変数をBP_GameStateに移行する
- 8.8 BP_GameStateに関数Initializeを作成し、初期化処理を置き換える
- 8.9 BP_JankenGameControllerの処理をBP_GameStateの変数を使用するように変更する
- 8.10 BP_GameStateのCallJankenStrに文字を設定する
- 8.11 BP_GameStateの変数をWBP_Gameのウィジェットにバインドする

Coreフォルダにブループリントを作成し、親クラスをGameStateに設定する

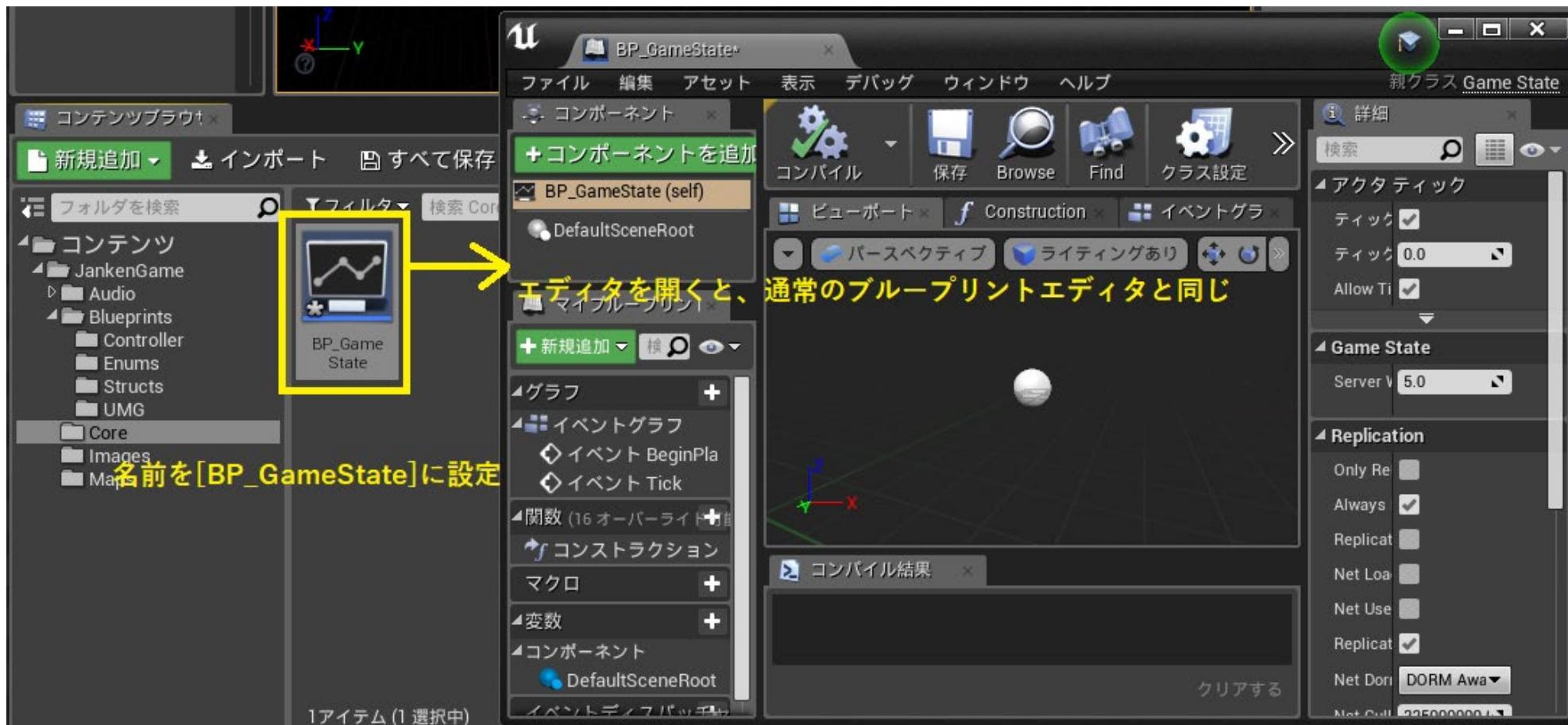


Coreフォルダを選択
> 右クリック
> ブループリント
> ブループリントクラス



すべてのクラス Gamestateで検索
> GameStateを選択
> 選択をクリック

名前をBP_GameStateに設定



重要なブループリントはCoreフォルダを使おう

2.5 重要なブループリントやその他のアセットのために、Core フォルダを使うこと lint unsupported

プロジェクトの動作に不可欠なアセットには /Content/Project/Core フォルダを使用してください。たとえば、
GameMode、Character、PlayerController、GameState、PlayerState、および関連するブループリントはここに属してい
なければなりません。

これは、他のチームメンバーにとって非常に明確な「これらに触れないでください」というメッセージになります。非エンジニアは `core` フォルダに入る理由はほとんどありません。優れたコード構造スタイルに従えば、デザイナーは、機能を公開する子クラスでゲームプレイを調整する必要があります。世界の建築家は、ベースクラスを潜在的に乱用するのではなく、指定されたフォルダでプレハブのブループリントを使用する必要があります。

たとえば、プロジェクトがレベルに配置できるピックアップを必要とする場合、ピックアップの基本動作を定義する基本ピックアップクラスが `core/Pickups` に存在するはずです。Health または Ammoなどの特定のピックアップは、`/Content/Project/Placeables/Pickups/` のようなフォルダに存在するはずです。ゲームデザイナーはこのフォルダ内のピックアップを定義し調整することができますが、プロジェクト全体のピックアップを意図せず破損する可能性があるため、`core/Pickups` に触れてはいけません。

UE4 Style Guide

2.5 重要なブループリントやその他のアセットのために、Core フォルダを使うこと

<http://bit.ly/2yU723S>

2e1 プロジェクトのコンテンツ構造の例

```
-- Content
  |-- GenericShooter
    |-- Art
      |-- Industrial
        |-- Ambient
        |-- Machinery
        |-- Pipes
      |-- Nature
        |-- Ambient
        |-- Foliage
        |-- Rocks
        |-- Trees
    |-- Office
  |-- Characters
    |-- Bob
    |-- Common
    |-- |-- Animations
    |-- |-- Audio
    |-- Jack
    |-- Steve
    |-- Zoe
  |-- Core
    |-- Characters
    |-- Engine
    |-- GameModes
    |-- Interactables
    |-- Pickups
    |-- Weapons
  |-- Effects
    |-- Electrical
```

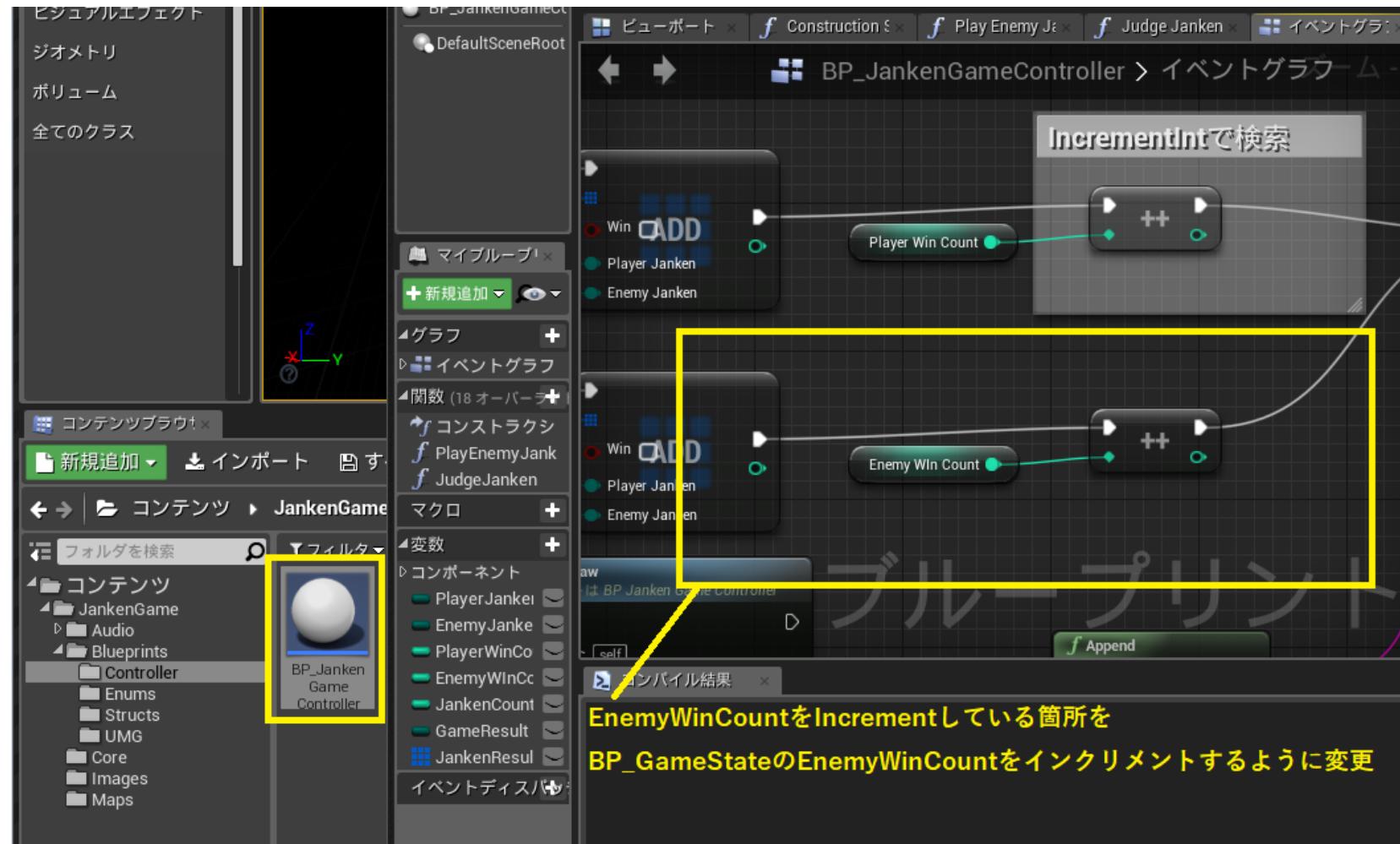
BP_GameStateに変数を追加



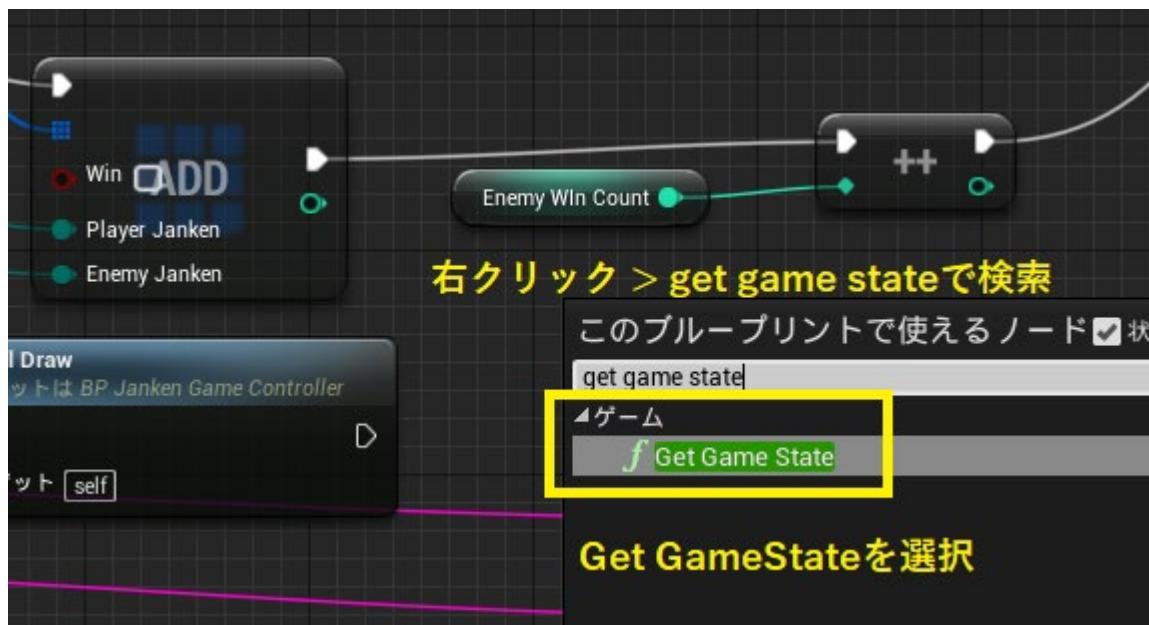
A screenshot of the Unreal Engine Blueprint Editor. On the left, there's a tree view with sections like '▲変数' (Variables) and '▲コンポーネント' (Components). Under 'Components', 'DefaultSceneRoot' is listed with a blue icon. In the main area, a variable named 'EnemyWinCount' is shown, highlighted with a yellow box. To its right is a small dropdown menu icon. At the top right of the main area, there's a yellow button with a plus sign and the text '変数' (Variable).

変数名	変数の型	デフォルト値
EnemyWinCount	Integer	-(設定なし)

BP_JankenGameControllerのEnemyWinCountを変更している処理を、
BP_GameStateのEnemyWinCountに置き換える



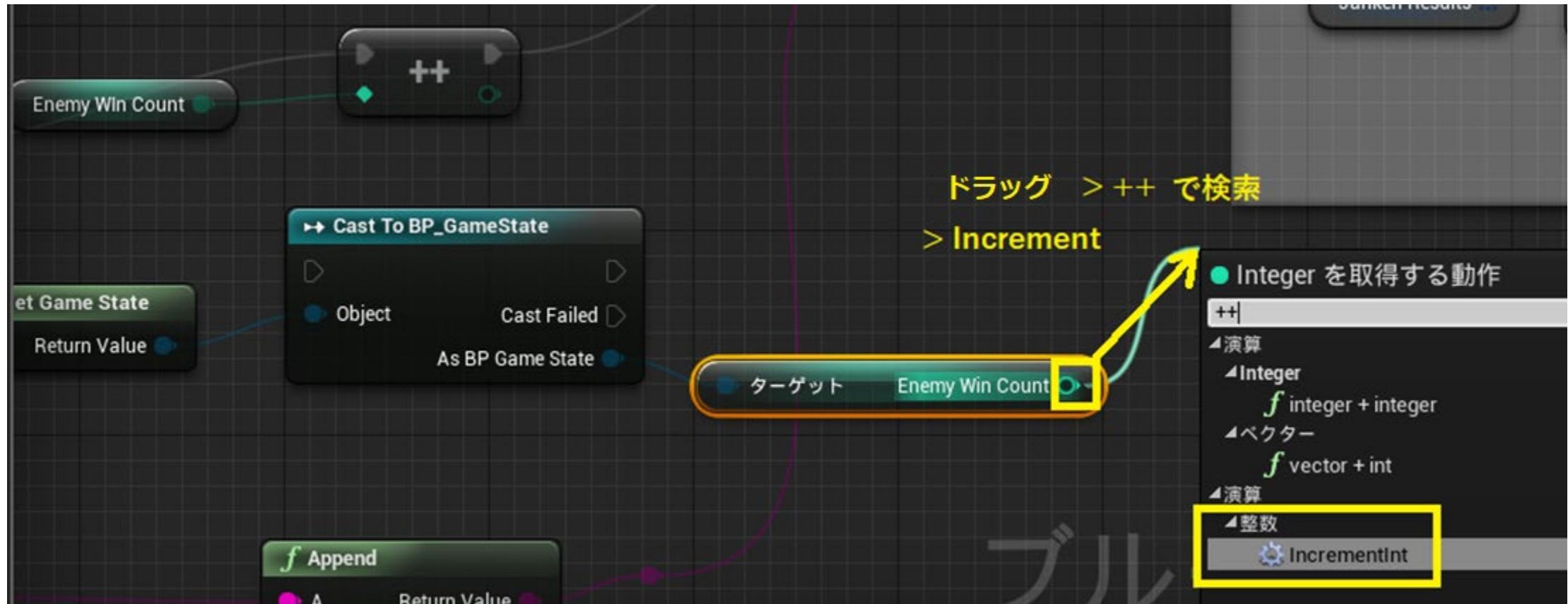
GetGameStateを追加する Cast To BP_GameStateを追加する



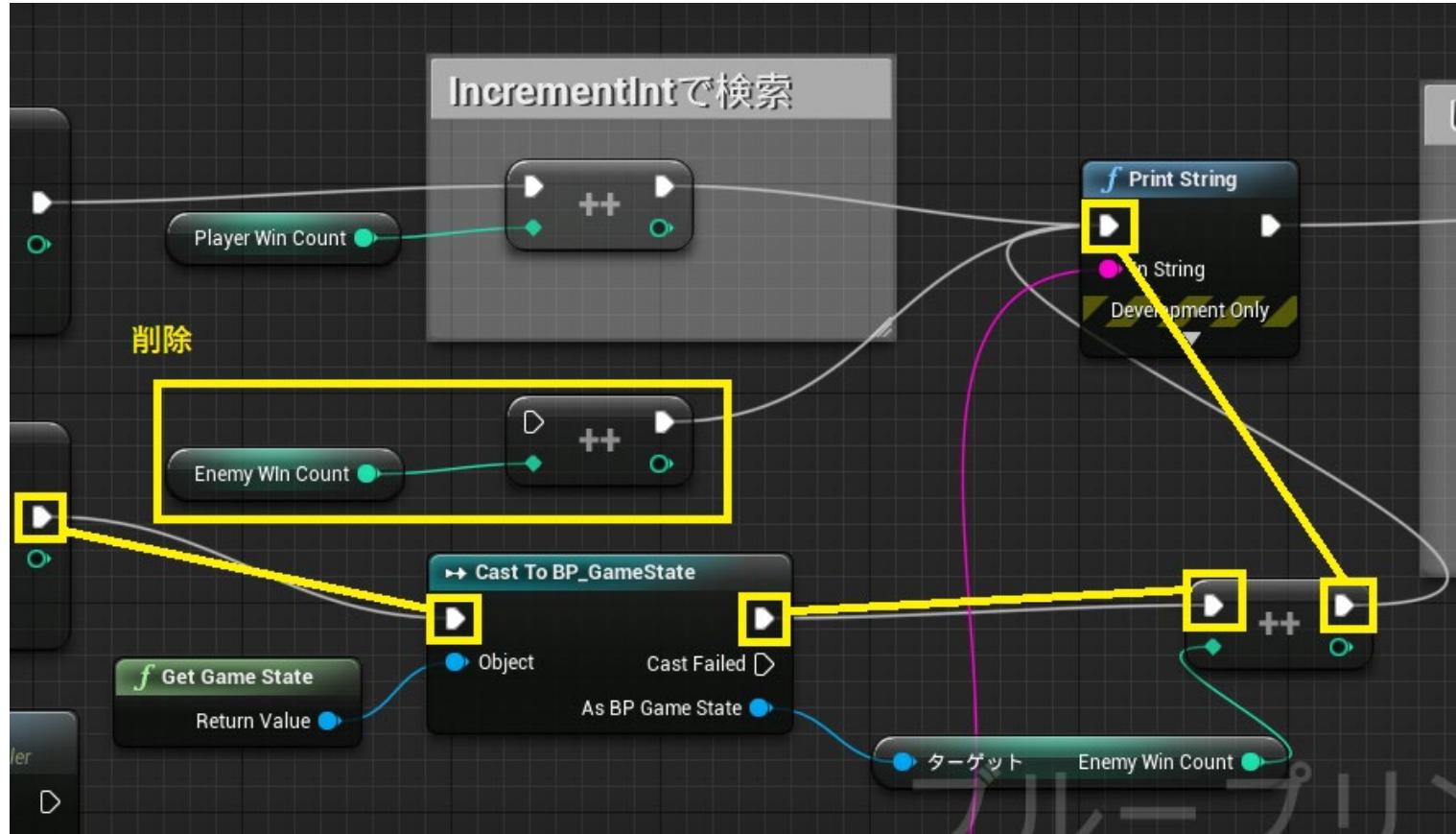
Cast To BP_GameState ノードの As BP Game State から
EnemyWinCount の Get を追加する



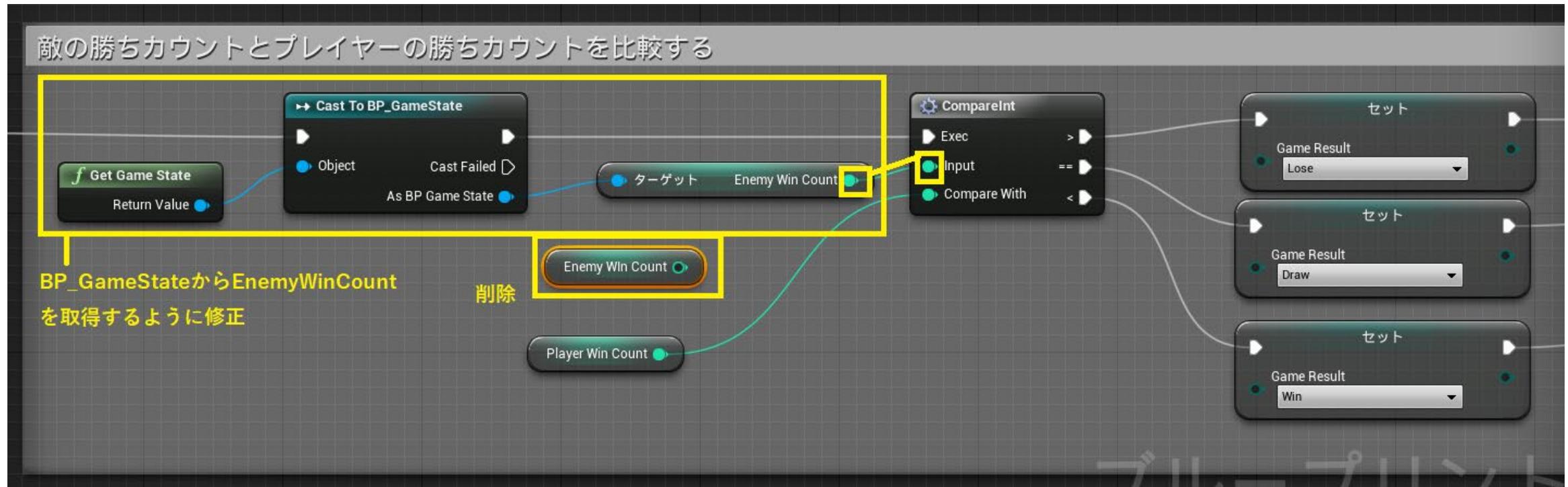
EnemyWinCountのGetからIncrementを追加する



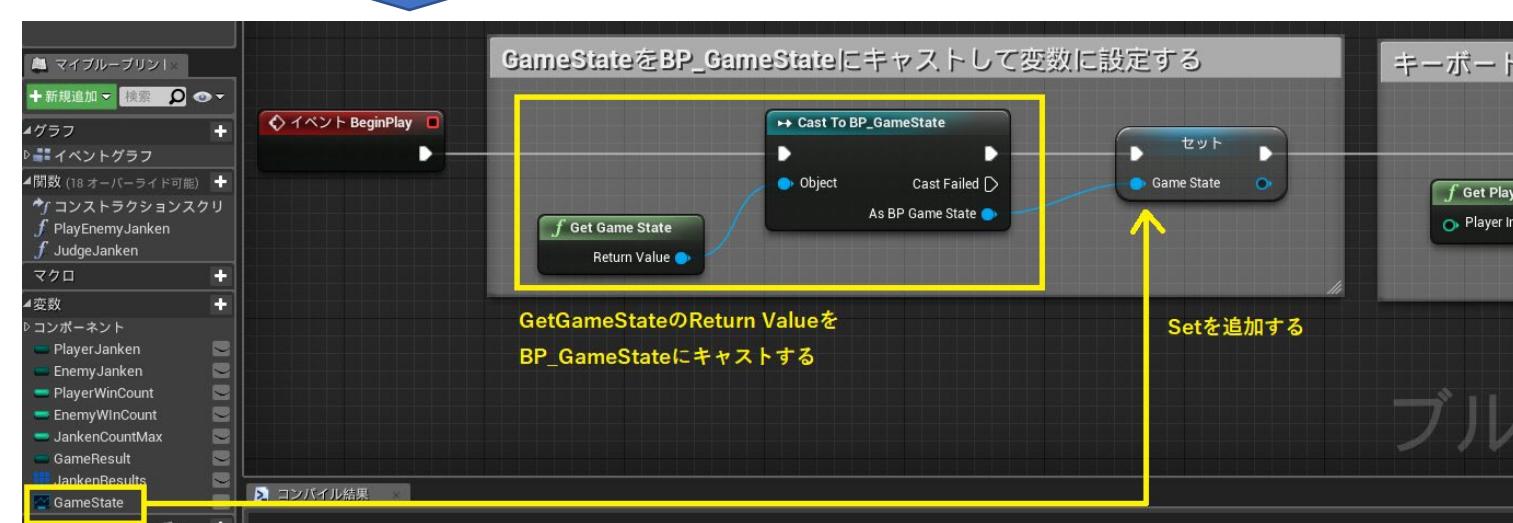
BP_GameStateの変数 Enemy Win CountのIncrementをするように処理を修正する



変数 EnemyWinCountを使用している箇所をBP_GameStateの変数
EnemyWinCountを使用するように変更

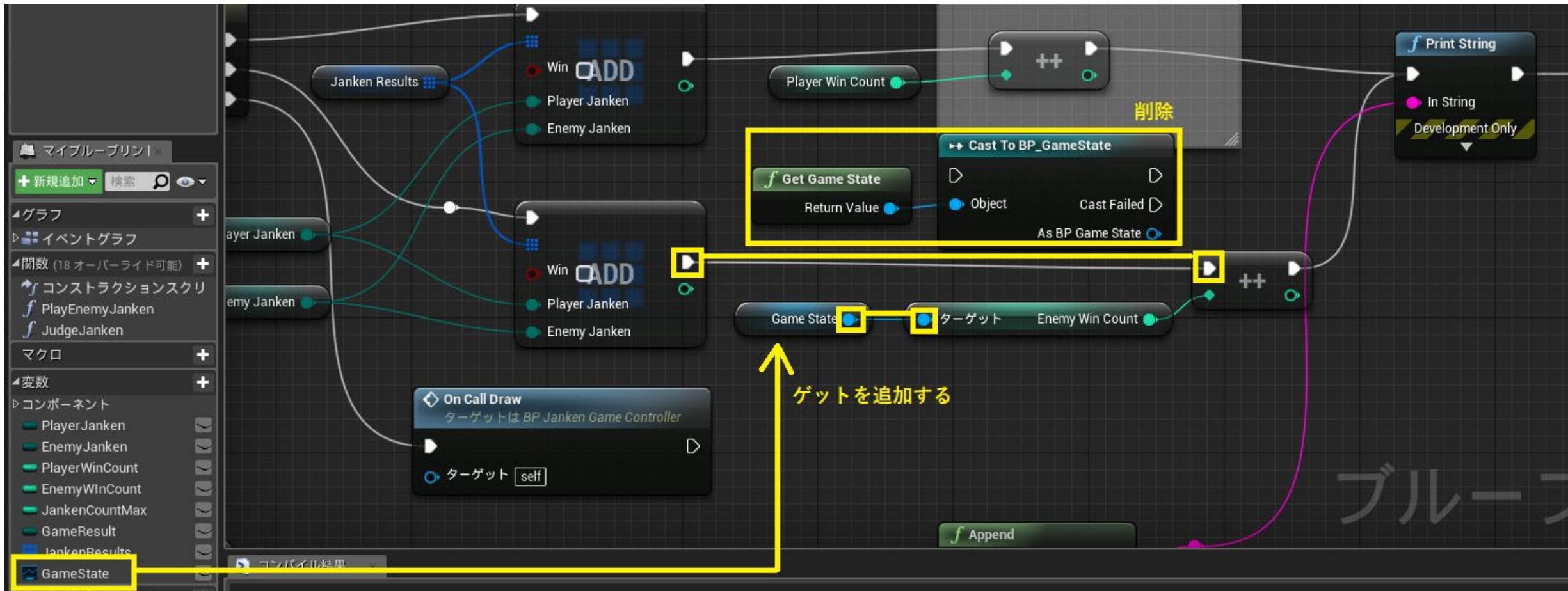


毎回 Castするのは面倒くさいので変数を用意し、BeginPlayの時に1度だけCastして変数に設定する

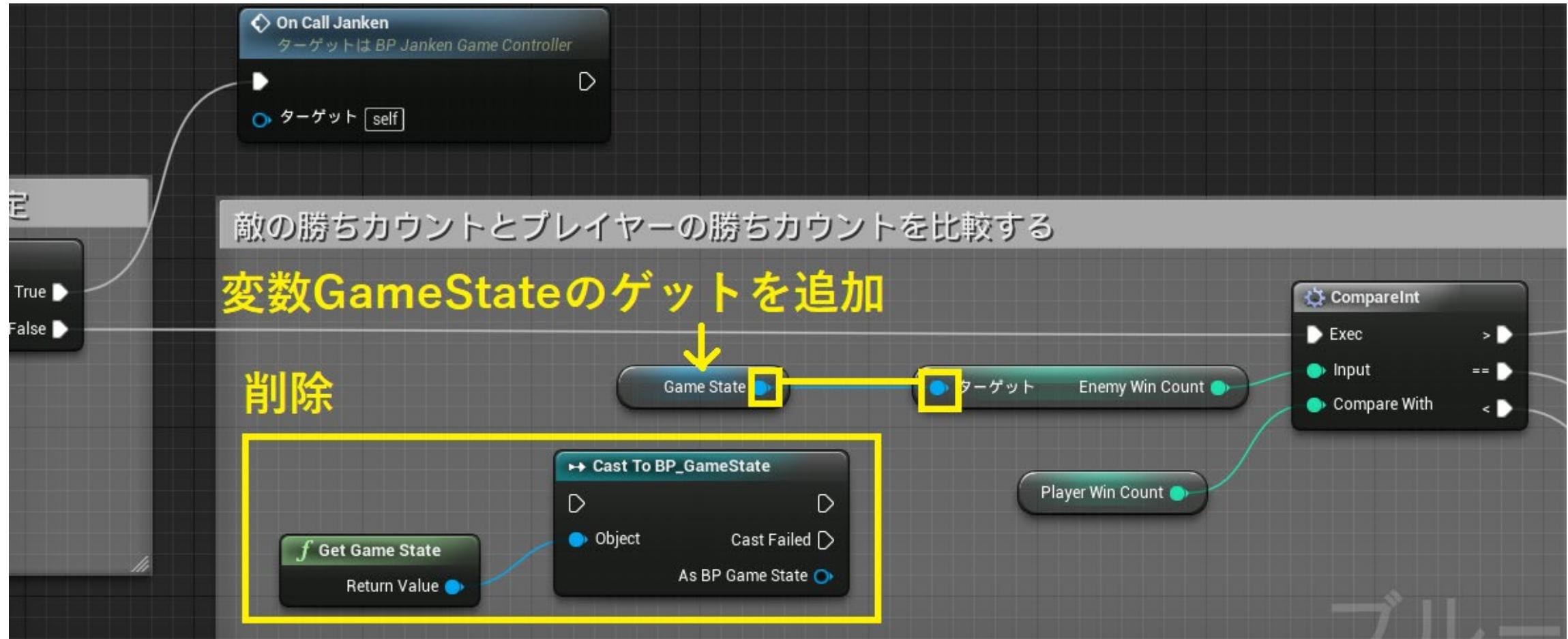


BeginPlay時にGameStateを
BP_GameStateにキャストした値を
変数に設定する

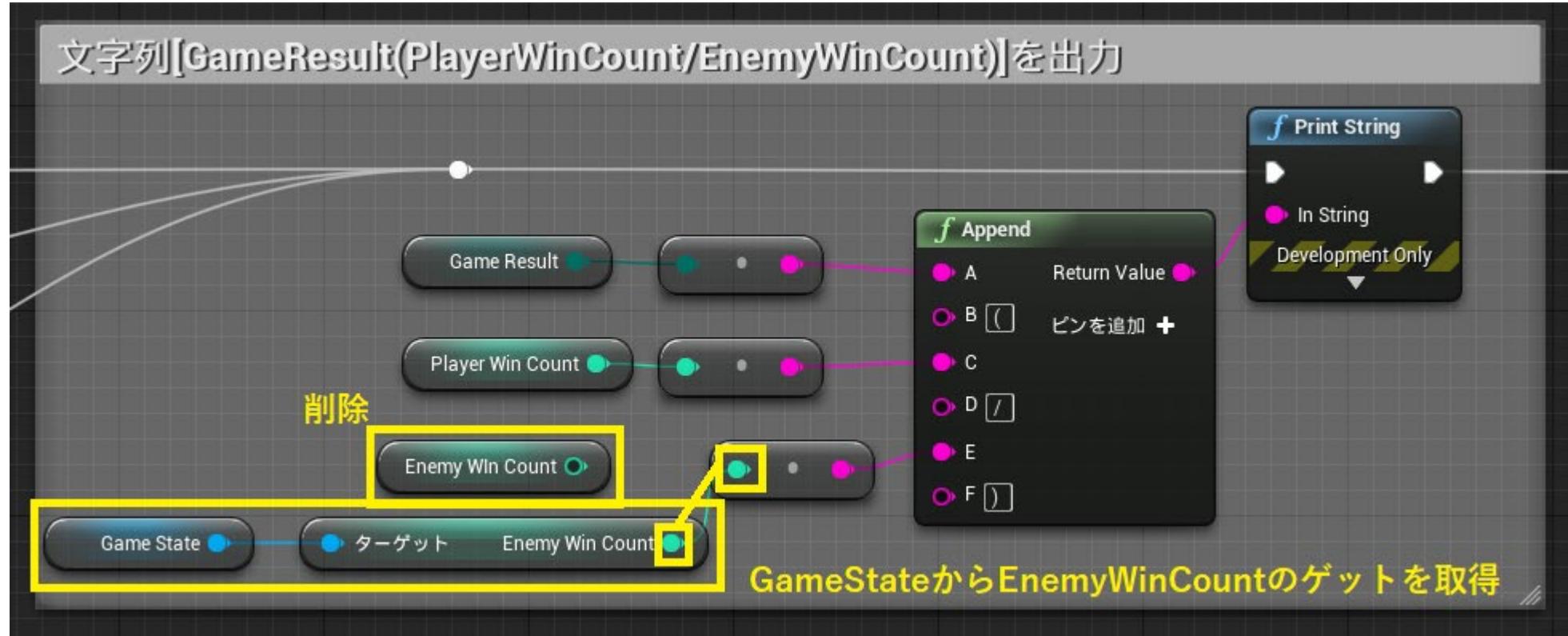
キャストしてBP_GameStateから取得しているEnemyWinCountを
変数のGameStateから取得するように修正



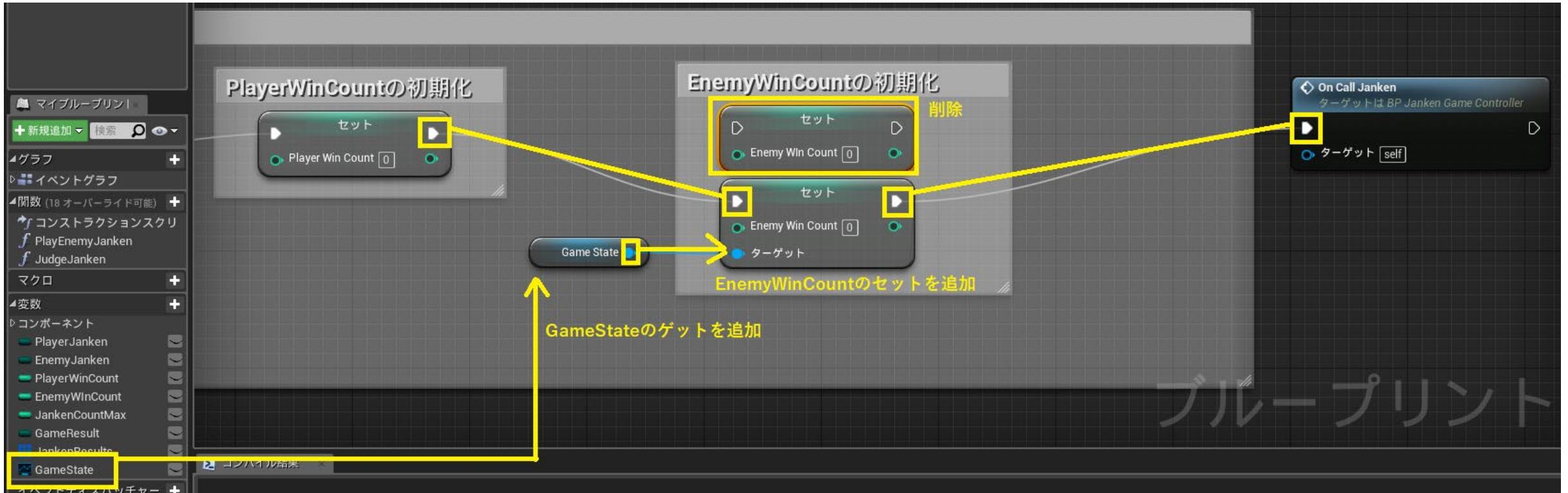
キャストしてBP_GameStateから取得しているEnemyWinCountを
変数のGameStateから取得するように修正



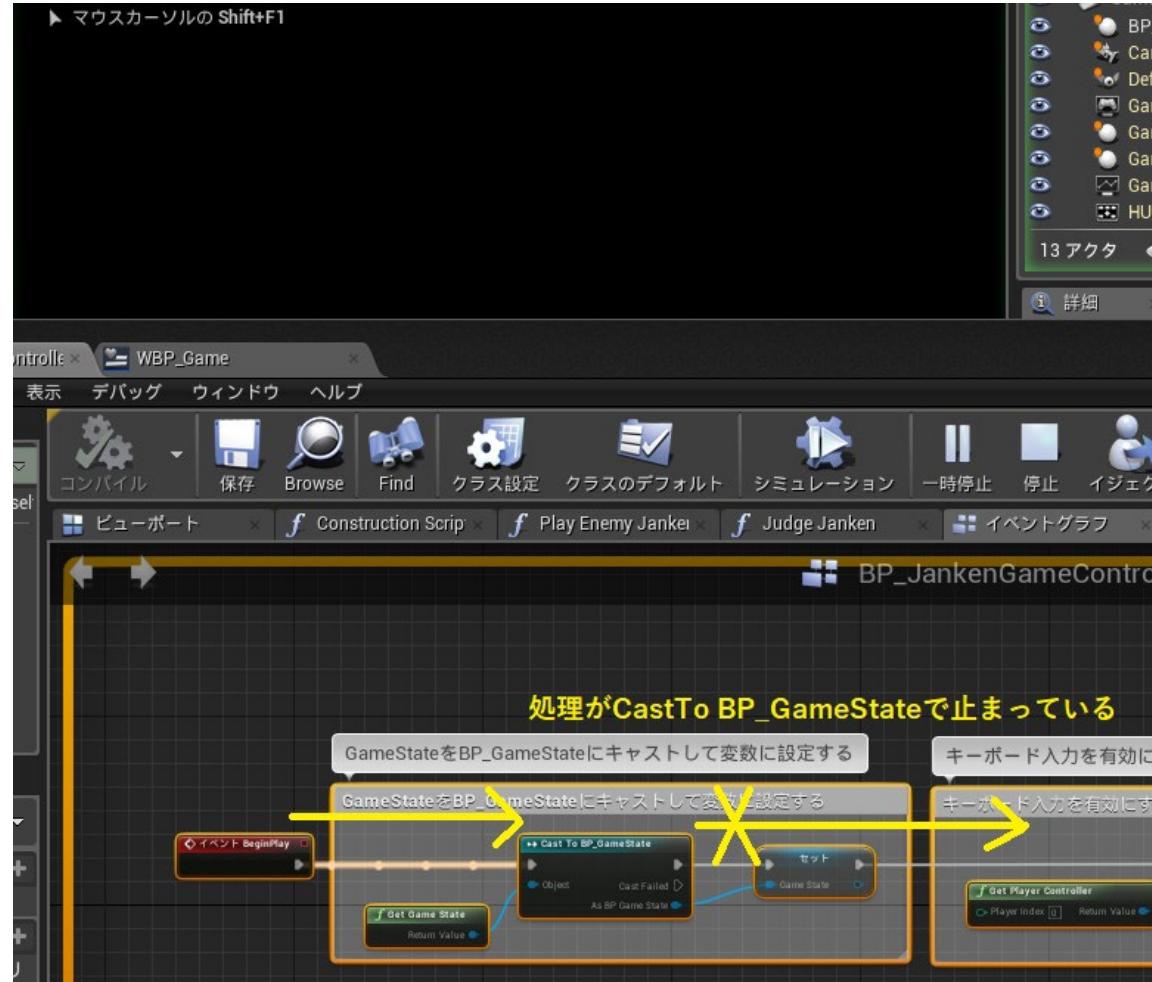
変数のGameStateからEnemyWinCountを取得するように修正



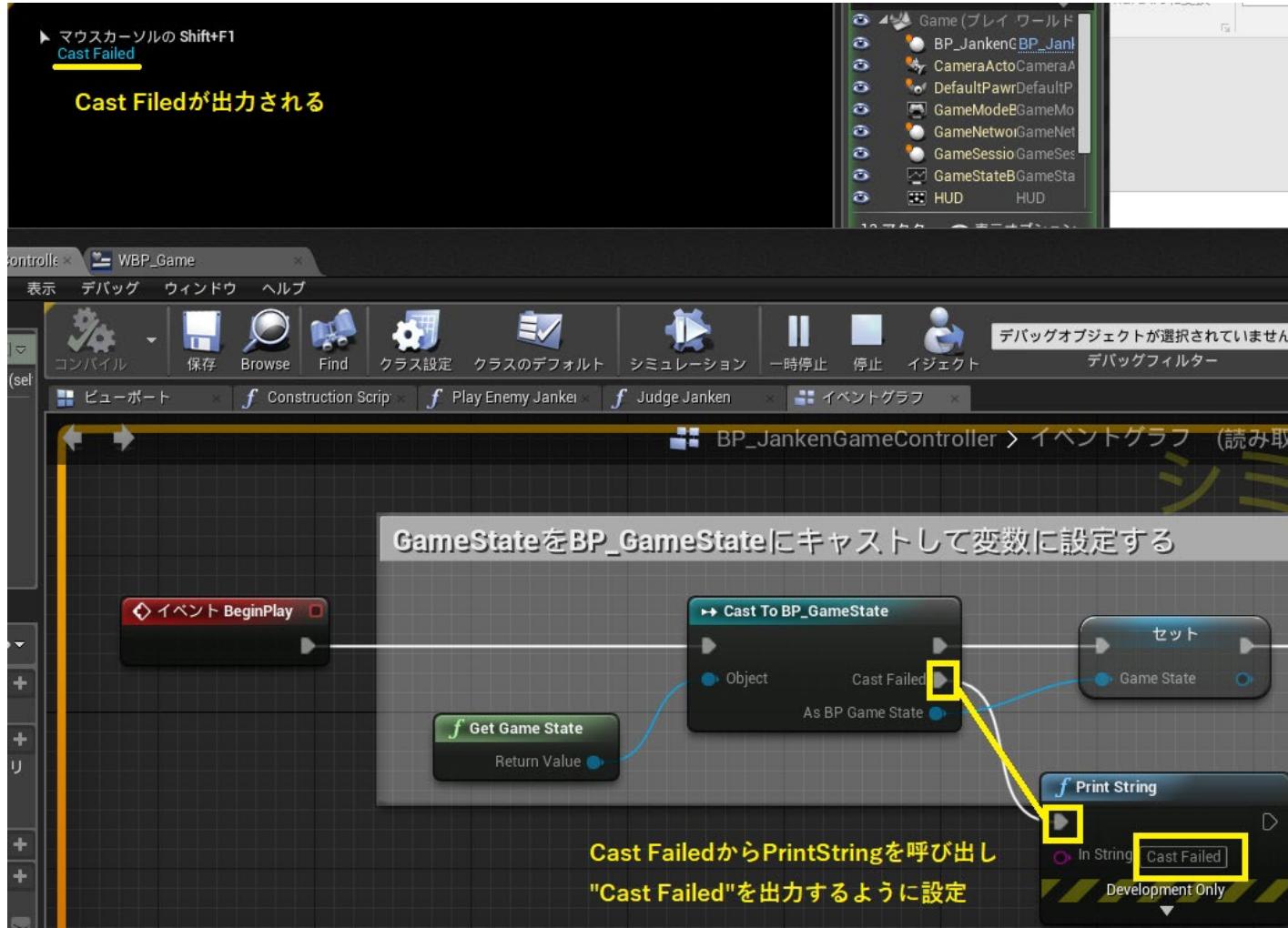
GameStateのEnemyWinCountを初期化するように修正



プレイして確認する
処理がCast To BP_GameStateで止まってしまう



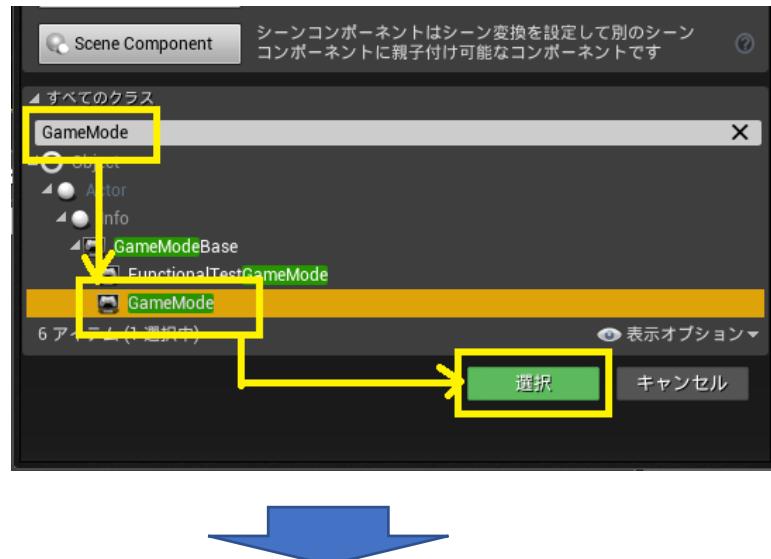
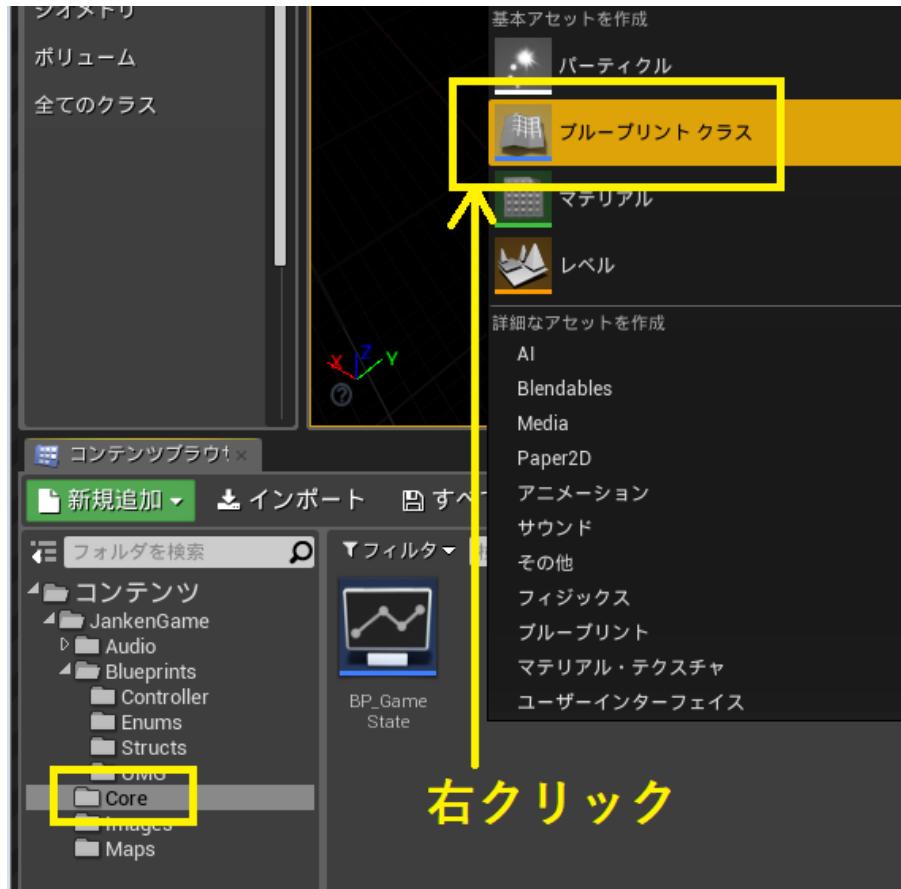
Cast To BP_GameStateからPrintStringで"Cast Failed"を出力するように設定



8. GameMode, GameStateを作成して、ゲームの状態を中継する

- 8.1 ウィジェットブループリントとゲームロジックの状態を中継する仕組みを作成する
- 8.2 BP_GameStateを作成する
- 8.3 BP_GameModeを作成する
- 8.4 WBP_Gameのウィジェットにバインドを作成して、BP_GameStateの値を反映させる
- 8.5 WBP_Gameを表示する
- 8.6 ゲームの状態を中継する仕組みの説明
- 8.7 BP_JankenGameControllerの変数をBP_GameStateに移行する
- 8.8 BP_GameStateに関数Initializeを作成し、初期化処理を置き換える
- 8.9 BP_JankenGameControllerの処理をBP_GameStateの変数を使用するように変更する
- 8.10 BP_GameStateのCallJankenStrに文字を設定する
- 8.11 BP_GameStateの変数をWBP_Gameのウィジェットにバインドする

ブループリント BP_GameMode(親クラス : GameMode)を作成

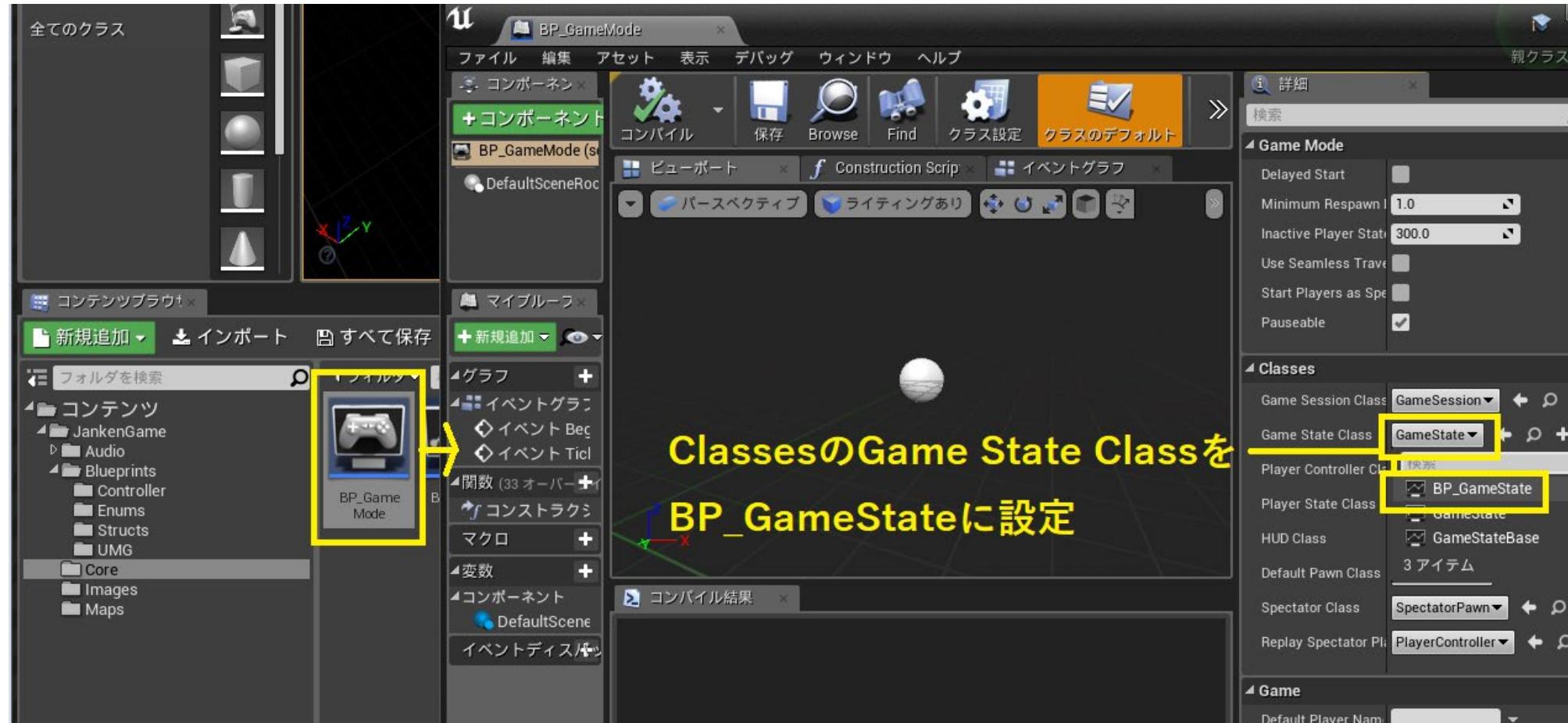


Core フォルダを選択
> 右クリック
> ブループリントクラス

GameModeで検索
> GameModeを選択
> 選択

名前を
BP_GameMode
に設定

BP_GameModeのGameStateClassにBP_GameStateを設定する

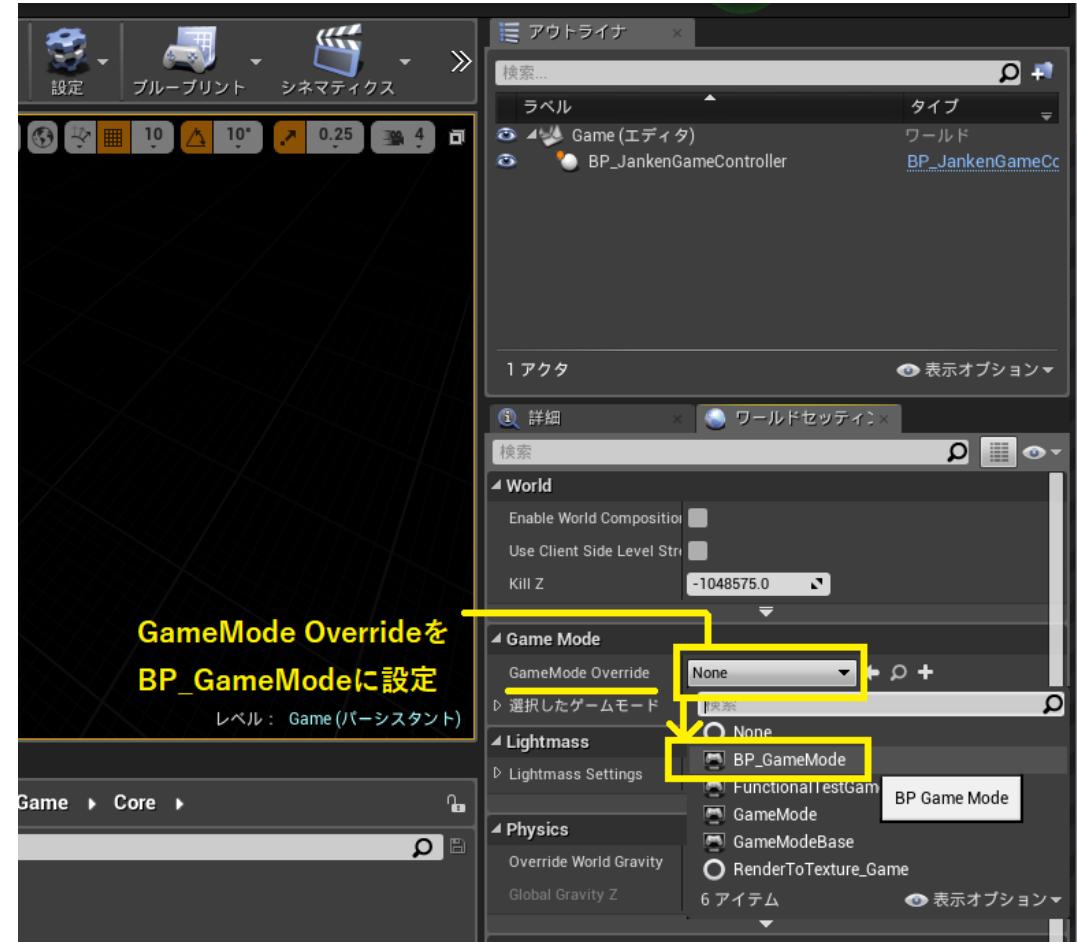


ワールドセッティングを表示する
ワールドセッティングのGameMode OverrideにBP_GameModeを設定

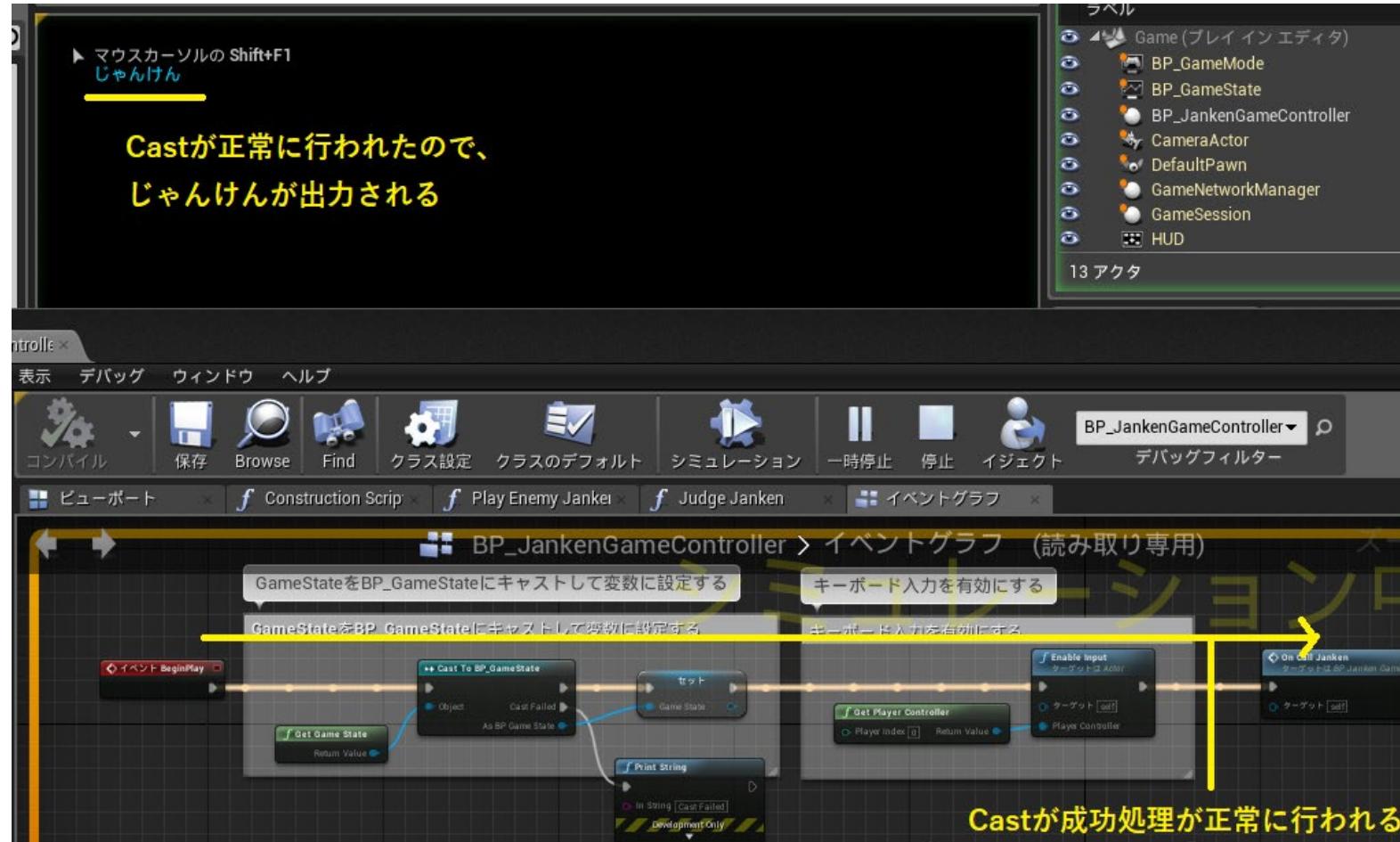


設定 > ワールドセッティング

GameMode Overrideを
BP_GameModeに設定



プレイして確認する
GameStateからBP_GameStateにCastが行われ処理が正常に行われる



8. GameMode, GameStateを作成して、ゲームの状態を中継する

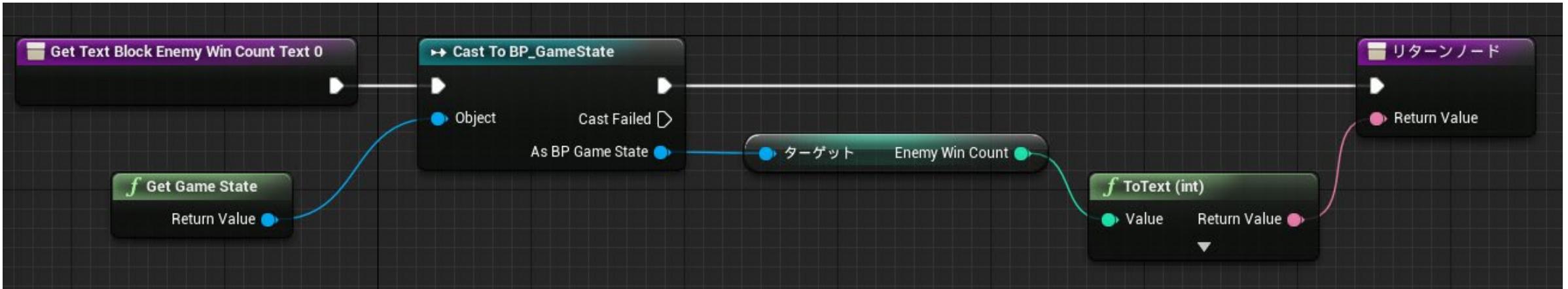
- 8.1 ウィジェットブループリントとゲームロジックの状態を中継する仕組みを作成する
- 8.2 BP_GameStateを作成する
- 8.3 BP_GameModeを作成する
- 8.4 WBP_Gameのウィジェットにバインドを作成して、BP_GameStateの値を反映させる
- 8.5 WBP_Gameを表示する
- 8.6 ゲームの状態を中継する仕組みの説明
- 8.7 BP_JankenGameControllerの変数をBP_GameStateに移行する
- 8.8 BP_GameStateに関数Initializeを作成し、初期化処理を置き換える
- 8.9 BP_JankenGameControllerの処理をBP_GameStateの変数を使用するように変更する
- 8.10 BP_GameStateのCallJankenStrに文字を設定する
- 8.11 BP_GameStateの変数をWBP_Gameのウィジェットにバインドする

WBP_GameのTextBlock_EnemyWinCountのTextにバインドを作成する



WBP_Gameを開く > TextBlock_EnemyWinCountを選択
> Textのバインドを作成

Textのバインド処理を実装
GameStateからEnemyWinCountを取得してTextの値に設定する



バインドは常にリターンノードに設定した値が反映される

BP_GameStateのEnemyWinCountが変更されたら、
TextBlock_EnemyWinCountのTextの表示が変更される

手順1

GetGameStateを追加

BP_GameStateのCastノードを追加



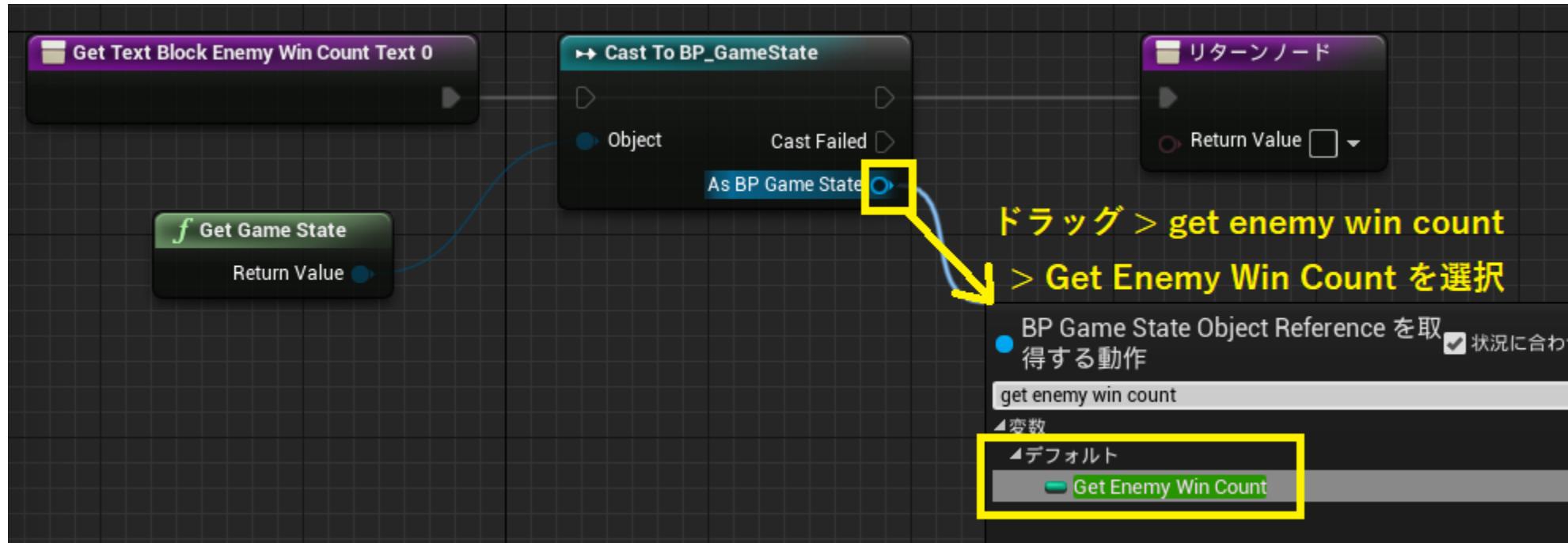
右クリック > get game stateで検索
> Get Game Stateを選択



Get Game StateのReturn Valueをドラッグ
> Castで検索
> Cast To BP_GameStateを選択

手順2

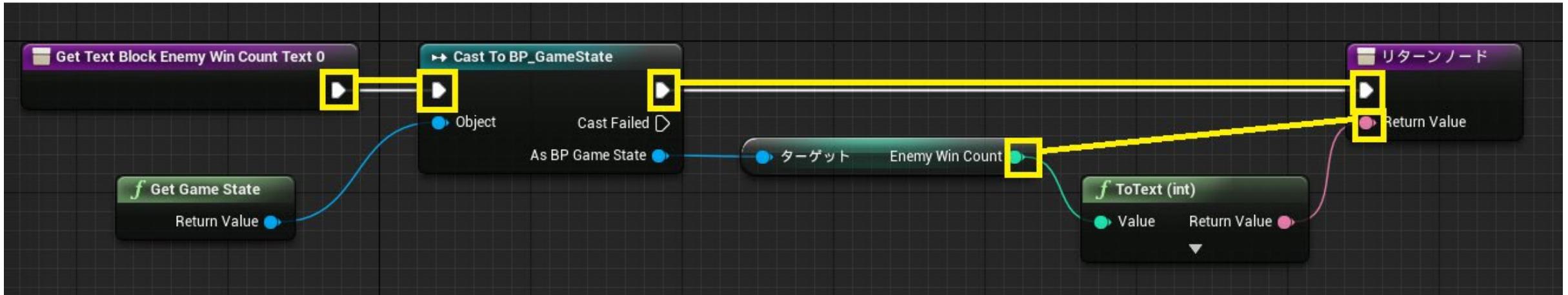
Cast To BP_GameStateのAs BP Game Stateから
Get Enemy Win Countを追加



手順3

BP_GameStateのEnemyWinCountの値をリターンノードのReturnValueに設定

EnemyWinCountからReturnValueに接続するとToText(int)ノードが自動的に追加される



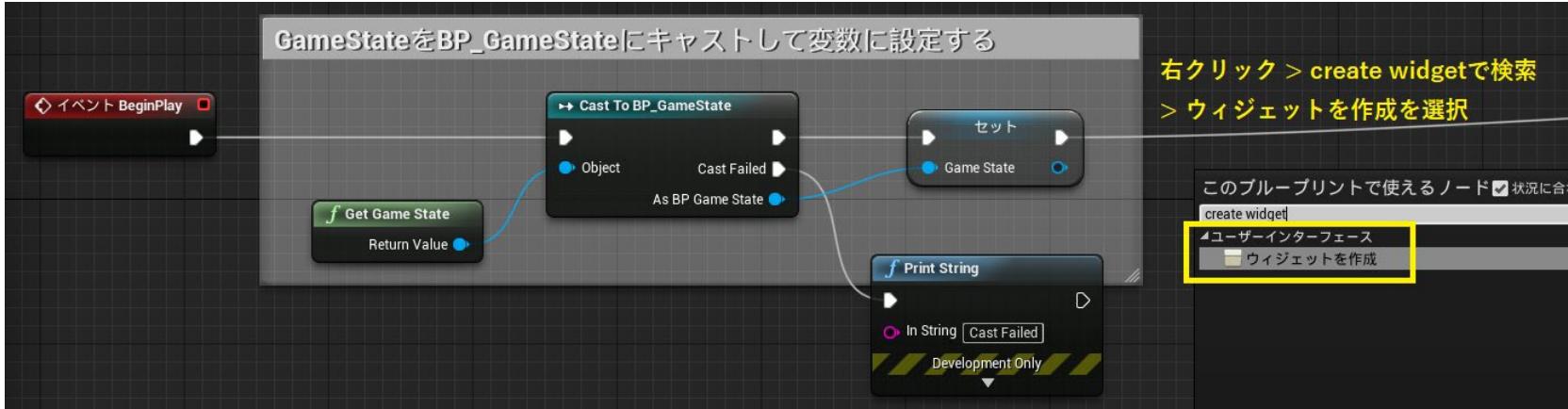
8. GameMode, GameStateを作成して、ゲームの状態を中継する

- 8.1 ウィジェットブループリントとゲームロジックの状態を中継する仕組みを作成する
- 8.2 BP_GameStateを作成する
- 8.3 BP_GameModeを作成する
- 8.4 WBP_Gameのウィジェットにバインドを作成して、BP_GameStateの値を反映させる
- 8.5 WBP_Gameを表示する
- 8.6 ゲームの状態を中継する仕組みの説明
- 8.7 BP_JankenGameControllerの変数をBP_GameStateに移行する
- 8.8 BP_GameStateに関数Initializeを作成し、初期化処理を置き換える
- 8.9 BP_JankenGameControllerの処理をBP_GameStateの変数を使用するように変更する
- 8.10 BP_GameStateのCallJankenStrに文字を設定する
- 8.11 BP_GameStateの変数をWBP_Gameのウィジェットにバインドする

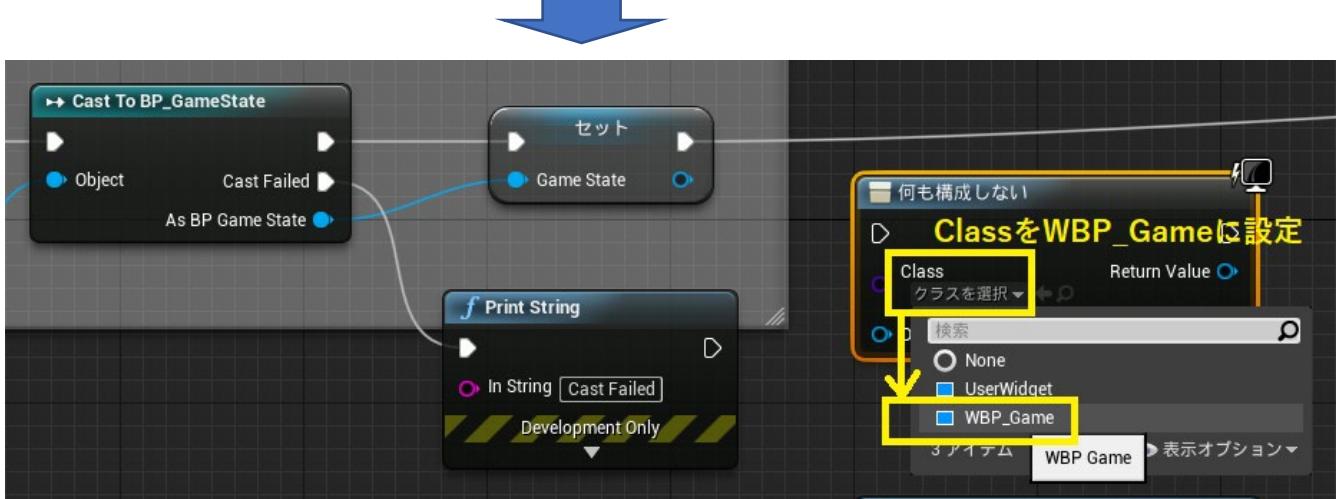
イベントBeginPlayにWBP_Gameを表示するように処理を追加する



ウィジェットを作成を追加 ClassをWBP_Gameに設定



右クリック
> Create widgeで検索
> ウィジェットを作成を選択

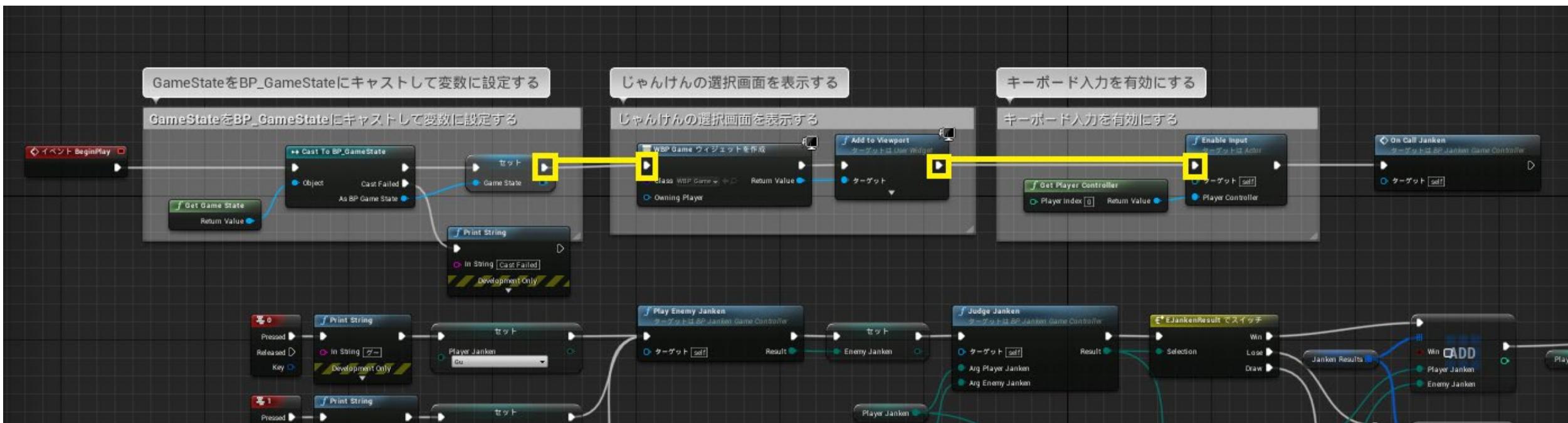


ウィジェットを作成のReturn ValueからAddToViewportを追加する



ウィジェットを作成のReturn Valueからドラッグ
> Add to viewportで検索
> Add to Viewportを選択

イベントBeginPlayにWBP_Gameを表示するように接続



プレイして確認する
負けた時に敵の勝ちカウントが+1される



8. GameMode, GameStateを作成して、ゲームの状態を中継する

- 8.1 ウィジェットブループリントとゲームロジックの状態を中継する仕組みを作成する
- 8.2 BP_GameStateを作成する
- 8.3 BP_GameModeを作成する
- 8.4 WBP_Gameのウィジェットにバインドを作成して、BP_GameStateの値を反映させる
- 8.5 WBP_Gameを表示する
- 8.6 ゲームの状態を中継する仕組みの説明
- 8.7 BP_JankenGameControllerの変数をBP_GameStateに移行する
- 8.8 BP_GameStateに関数Initializeを作成し、初期化処理を置き換える
- 8.9 BP_JankenGameControllerの処理をBP_GameStateの変数を使用するように変更する
- 8.10 BP_GameStateのCallJankenStrに文字を設定する
- 8.11 BP_GameStateの変数をWBP_Gameのウィジェットにバインドする

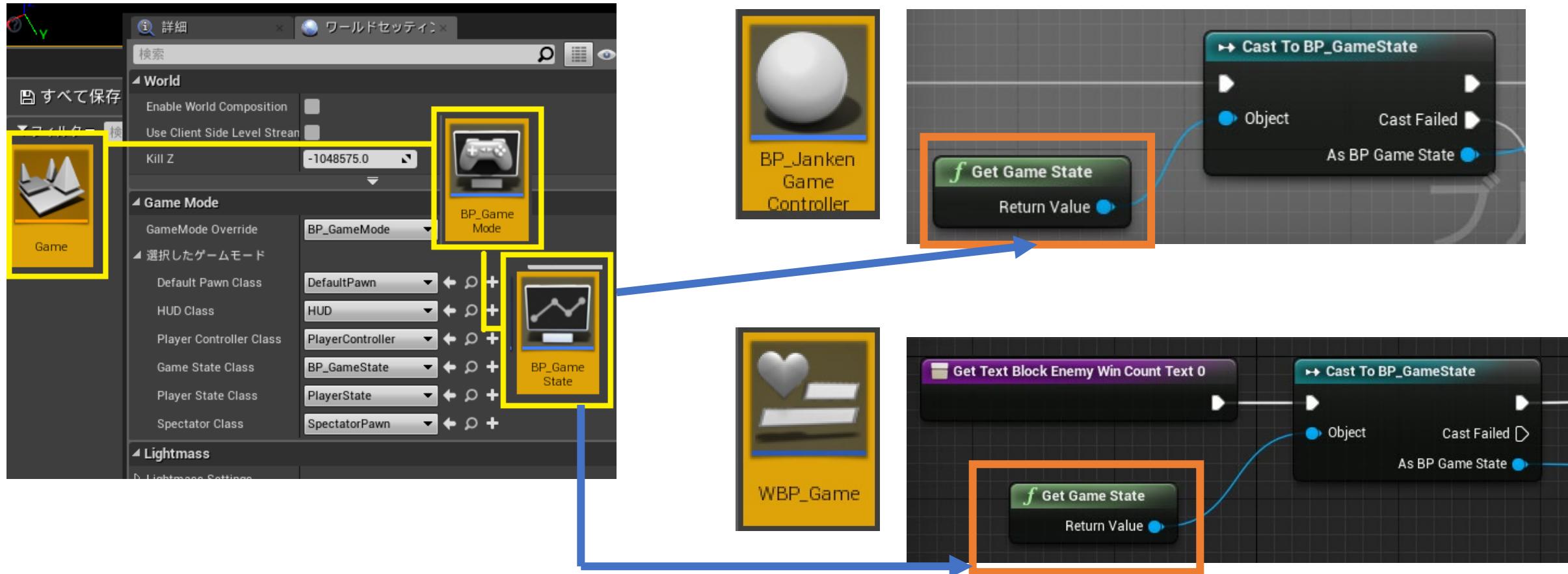
レベルにGameModeを設定する GameModeにGameStateを設定する



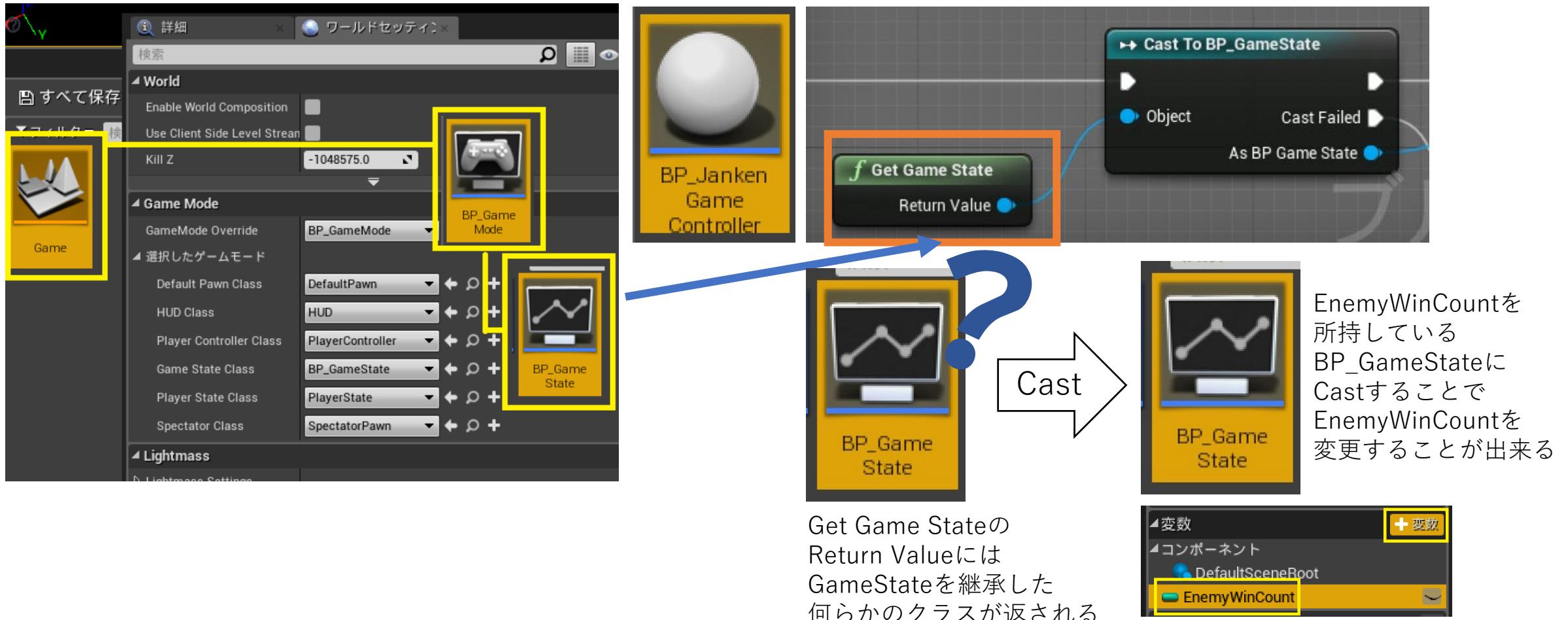
全て1対1の関係

関係性の強さは
レベル > GameMode > GameState

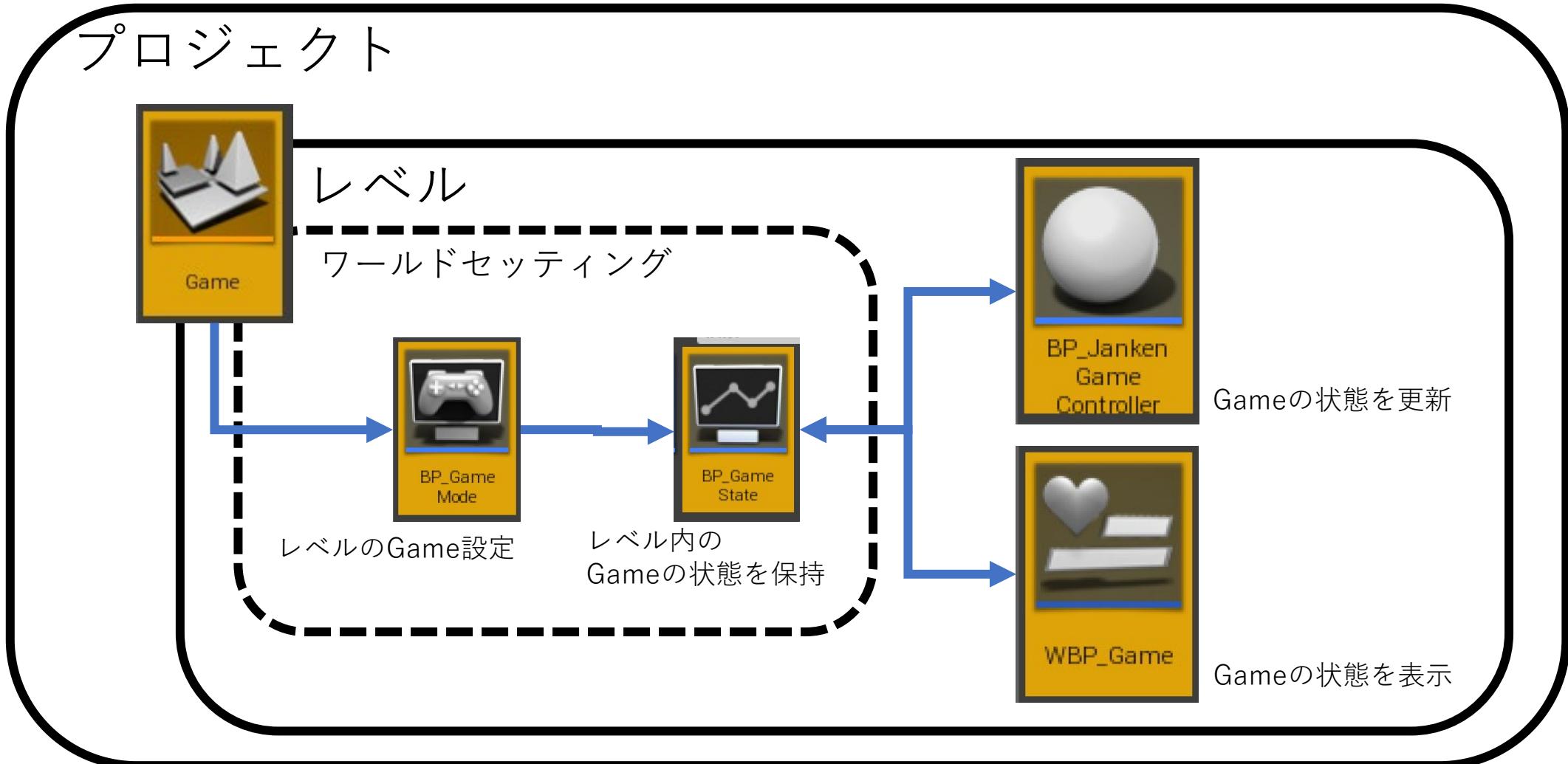
ブループリントからGetGameStateノードで
GameModeに設定したGameStateを取得することが出来る



GetGameStateノードのReturnValueにはGameStateを継承したクラスが設定されている
EnemyWinCountを所持している、BP_GameStateにCastすることで、EnemyWinCountを変更することが出来る



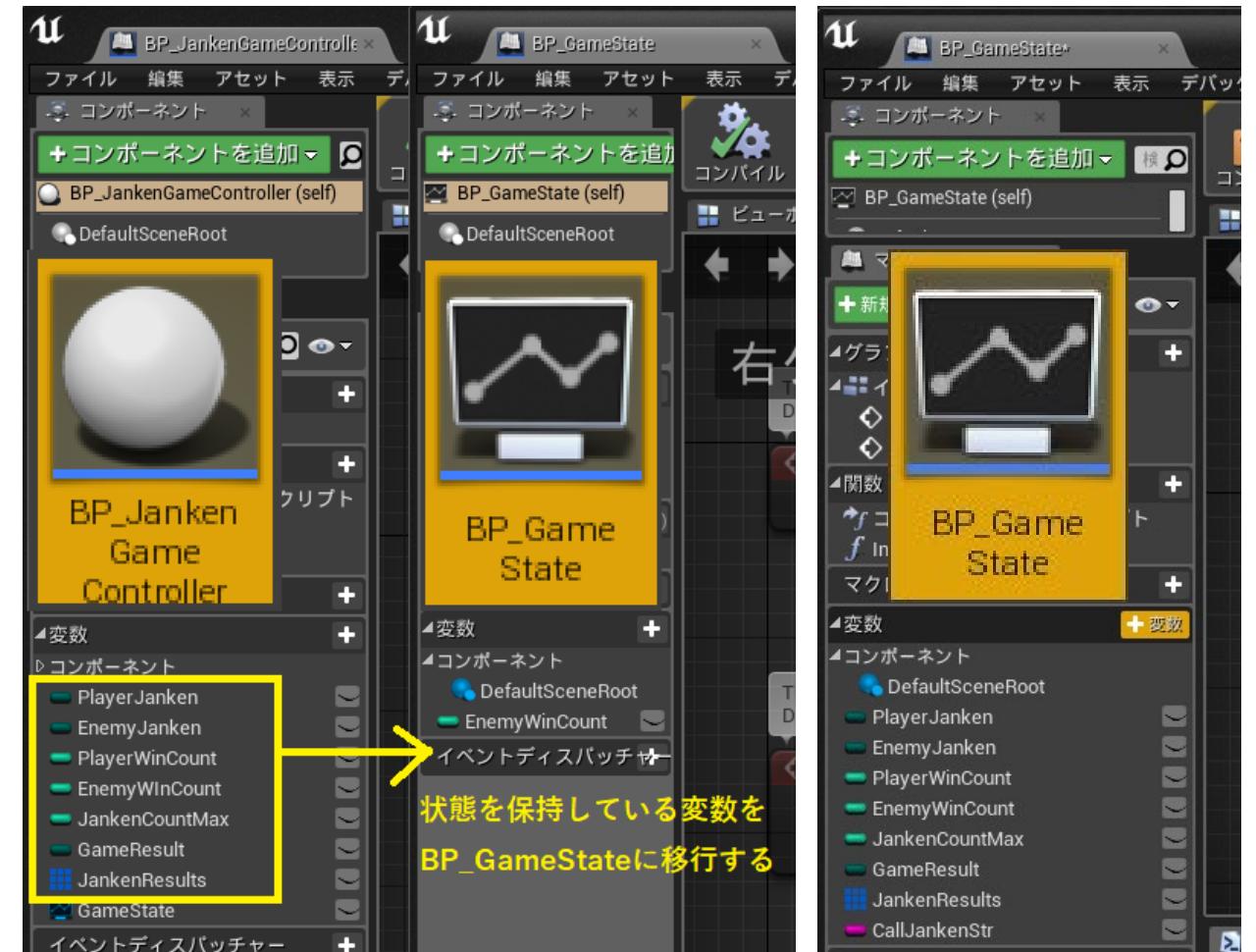
関連性を図に表してみる



8. GameMode, GameStateを作成して、ゲームの状態を中継する

- 8.1 ウィジェットブループリントとゲームロジックの状態を中継する仕組みを作成する
- 8.2 BP_GameStateを作成する
- 8.3 BP_GameModeを作成する
- 8.4 WBP_Gameのウィジェットにバインドを作成して、BP_GameStateの値を反映させる
- 8.5 WBP_Gameを表示する
- 8.6 ゲームの状態を中継する仕組みの説明
- 8.7 BP_JankenGameControllerの変数をBP_GameStateに移行する
- 8.8 BP_GameStateに関数Initializeを作成し、初期化処理を置き換える
- 8.9 BP_JankenGameControllerの処理をBP_GameStateの変数を使用するように変更する
- 8.10 BP_GameStateのCallJankenStrに文字を設定する
- 8.11 BP_GameStateの変数をWBP_Gameのウィジェットにバインドする

BP_JankenGameControllerの変数を BP_GameStateに移行する



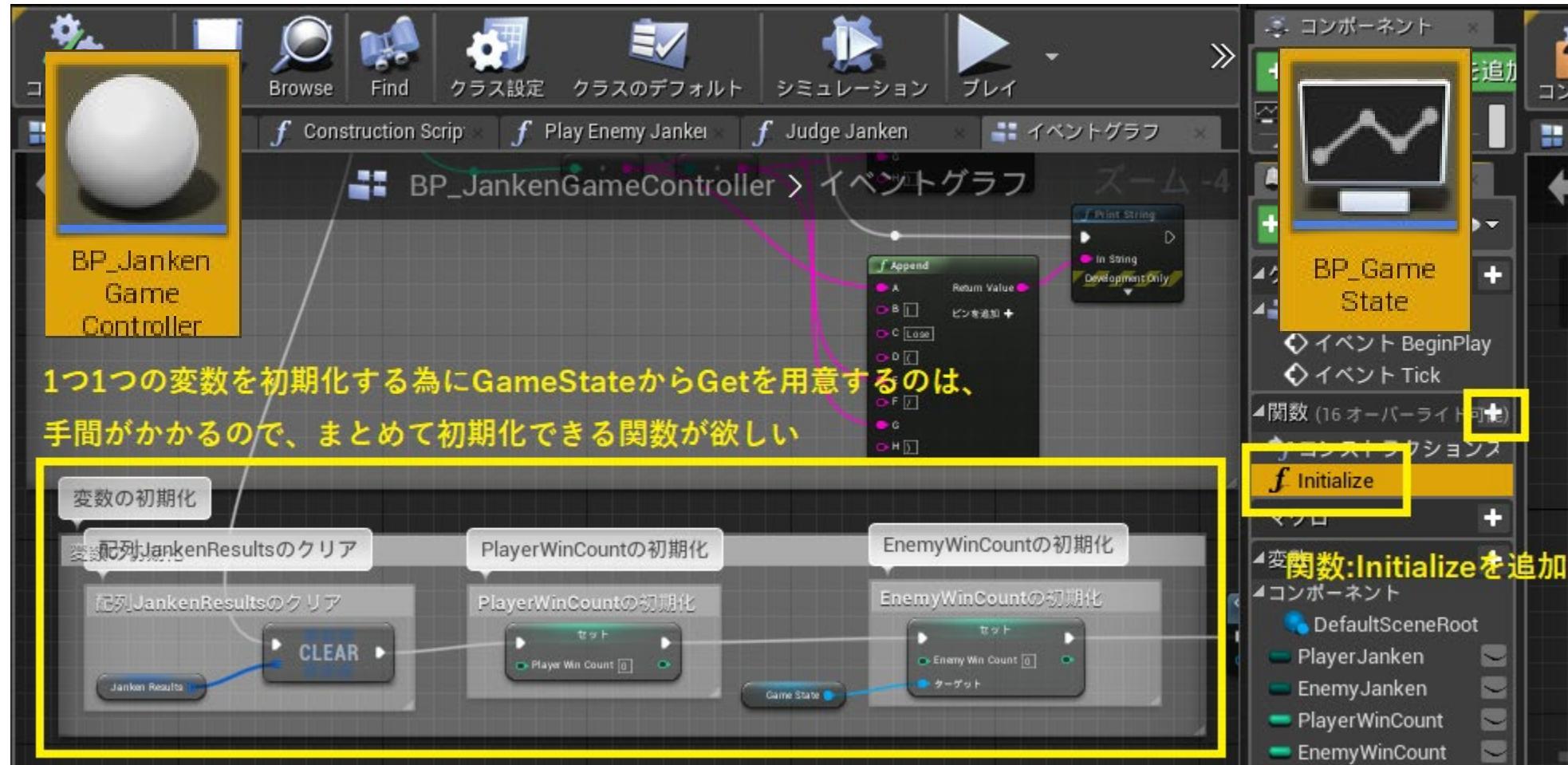
変数名	変数の型	デフォルト値
PlayerJanken	EJanken	-(設定なし)
EnemyJanken	EJanken	-(設定なし)
PlayerWinCount	Integer	-(設定なし)
EnemyWinCount	Integer	-(設定なし)
JankenCountMax	Integer	5
GameResult	EJankenResult	-(設定なし)
JankenResults	FJankenResult (配列)	-(設定なし)
CallJankenStr	String	-(設定なし)

CallJankenStrは[じゃんけん]や[あいこ]の掛け声を表示するために使用するため追加

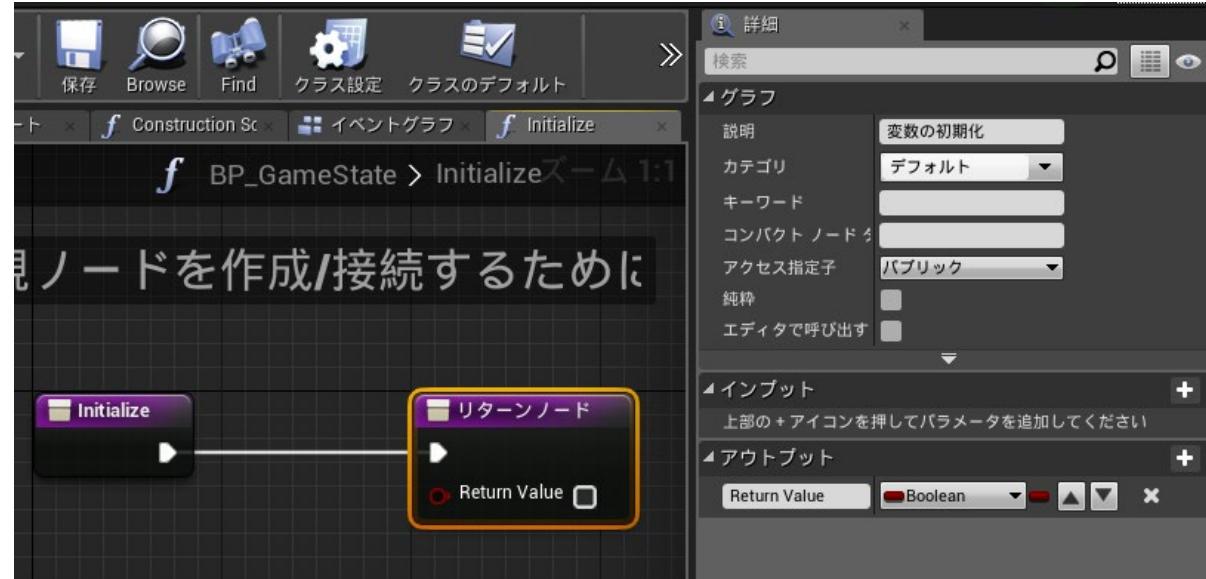
8. GameMode, GameStateを作成して、ゲームの状態を中継する

- 8.1 ウィジェットブループリントとゲームロジックの状態を中継する仕組みを作成する
- 8.2 BP_GameStateを作成する
- 8.3 BP_GameModeを作成する
- 8.4 WBP_Gameのウィジェットにバインドを作成して、BP_GameStateの値を反映させる
- 8.5 WBP_Gameを表示する
- 8.6 ゲームの状態を中継する仕組みの説明
- 8.7 BP_JankenGameControllerの変数をBP_GameStateに移行する
- 8.8 BP_GameStateに関数Initializeを作成し、初期化処理を置き換える
- 8.9 BP_JankenGameControllerの処理をBP_GameStateの変数を使用するように変更する
- 8.10 BP_GameStateのCallJankenStrに文字を設定する
- 8.11 BP_GameStateの変数をWBP_Gameのウィジェットにバインドする

BP_GameStateに関数Initializeを追加する



説明、アウトプットを設定する



見ノードを作成/接続するために

3.3.2 すべての関数にReturnノードが必須

<http://bit.ly/2ie3krs>

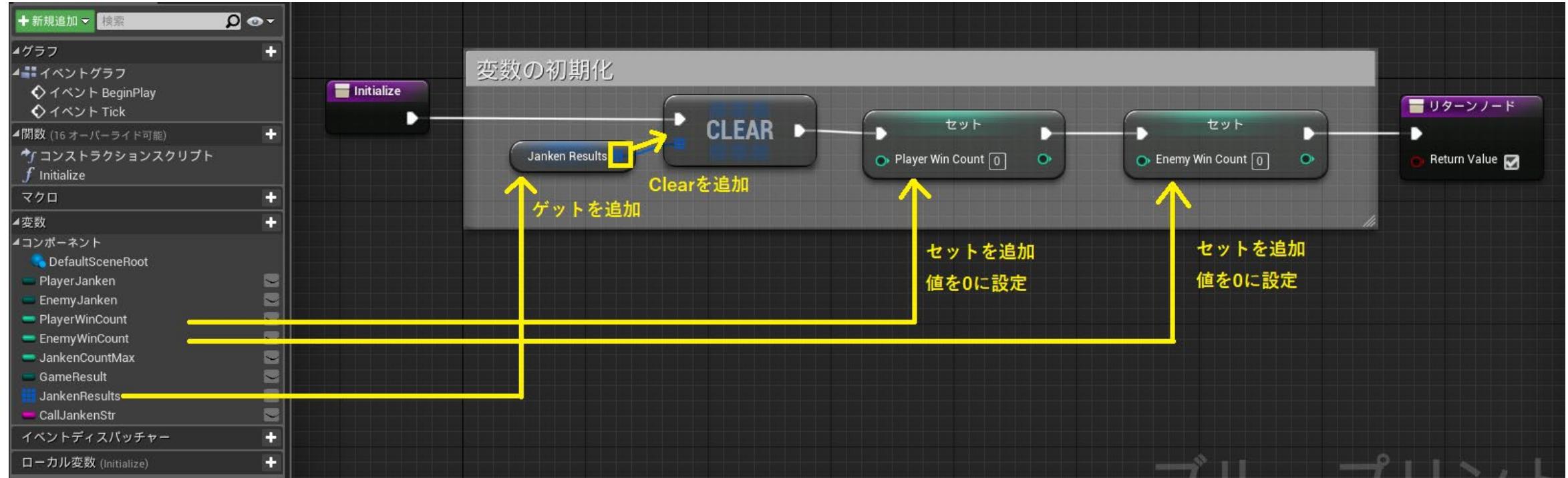
3.3.4 全てのPublic関数には説明文を記載するべき

<http://bit.ly/2jqhHMp>

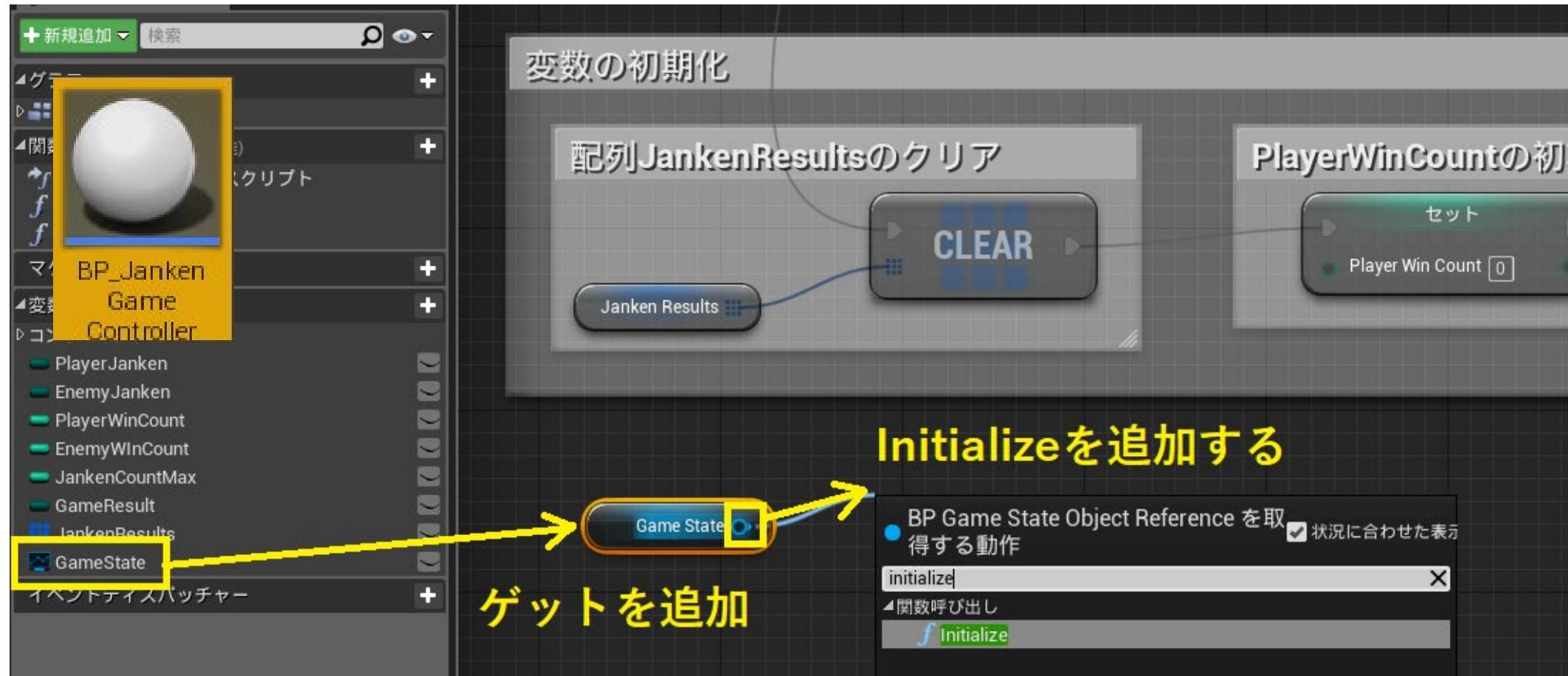
関数名	アクセス指定子	Input/Output	パラメータ名	型
Initialize	パブリック	Output	Return Value	Boolean

説明 变数の初期化

変数を初期化する

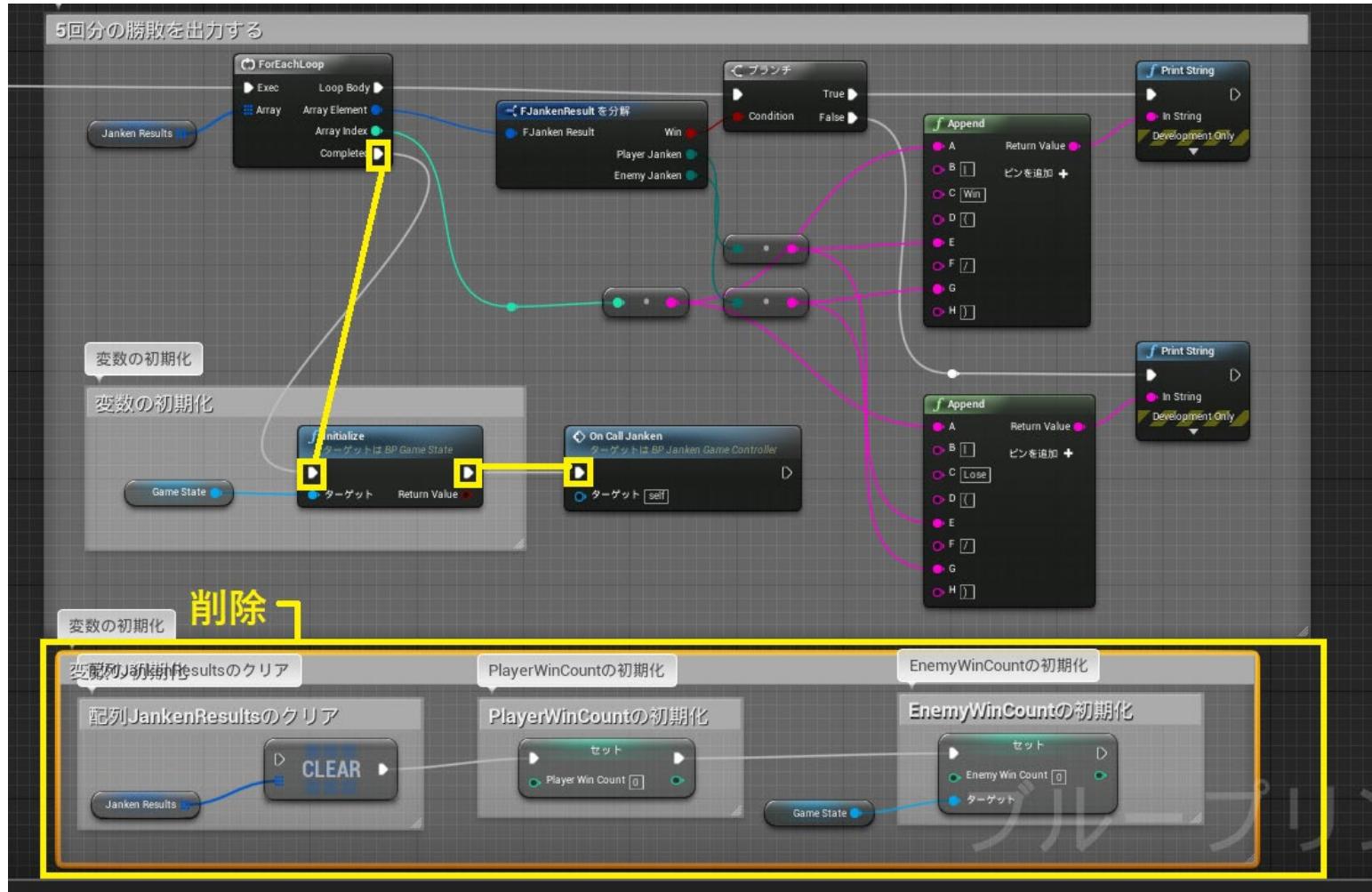


BP_JankenGameControllerの変数の初期化処理をしている箇所を、
GameStateのInitializeを呼ぶように置き換える 1



変数GameStateのゲットからInitializeを追加する

BP_JankenGameControllerの変数の初期化処理をしている箇所を、GameStateのInitializeを呼ぶように置き換える 2

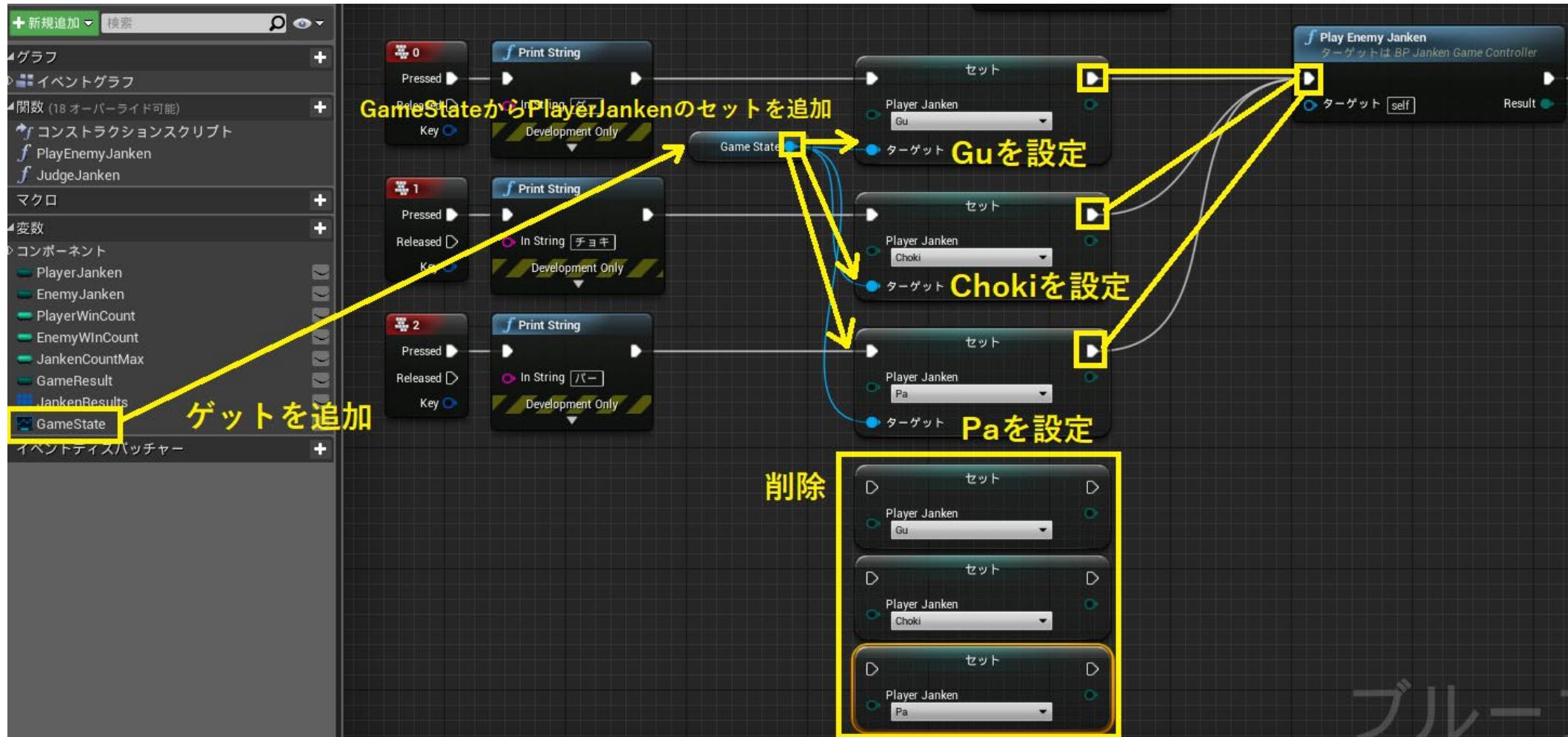


ForEachLoopのCompletedから
Initializeを呼び出すように、初期化
処理を置き換える
不要になった初期化処理は削除する

8. GameMode, GameStateを作成して、ゲームの状態を中継する

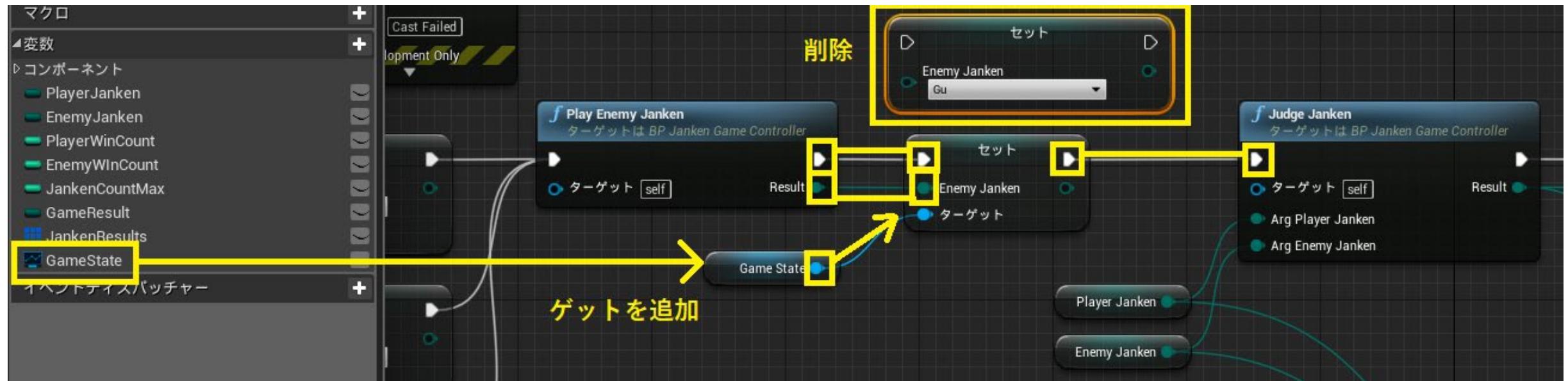
- 8.1 ウィジェットブループリントとゲームロジックの状態を中継する仕組みを作成する
- 8.2 BP_GameStateを作成する
- 8.3 BP_GameModeを作成する
- 8.4 WBP_Gameのウィジェットにバインドを作成して、BP_GameStateの値を反映させる
- 8.5 WBP_Gameを表示する
- 8.6 ゲームの状態を中継する仕組みの説明
- 8.7 BP_JankenGameControllerの変数をBP_GameStateに移行する
- 8.8 BP_GameStateに関数Initializeを作成し、初期化処理を置き換える
- 8.9 BP_JankenGameControllerの処理をBP_GameStateの変数を使用するように変更する
- 8.10 BP_GameStateのCallJankenStrに文字を設定する
- 8.11 BP_GameStateの変数をWBP_Gameのウィジェットにバインドする

BP_JankenGameControllerの処理をBP_GameStateの 変数を使用するように変更する 1

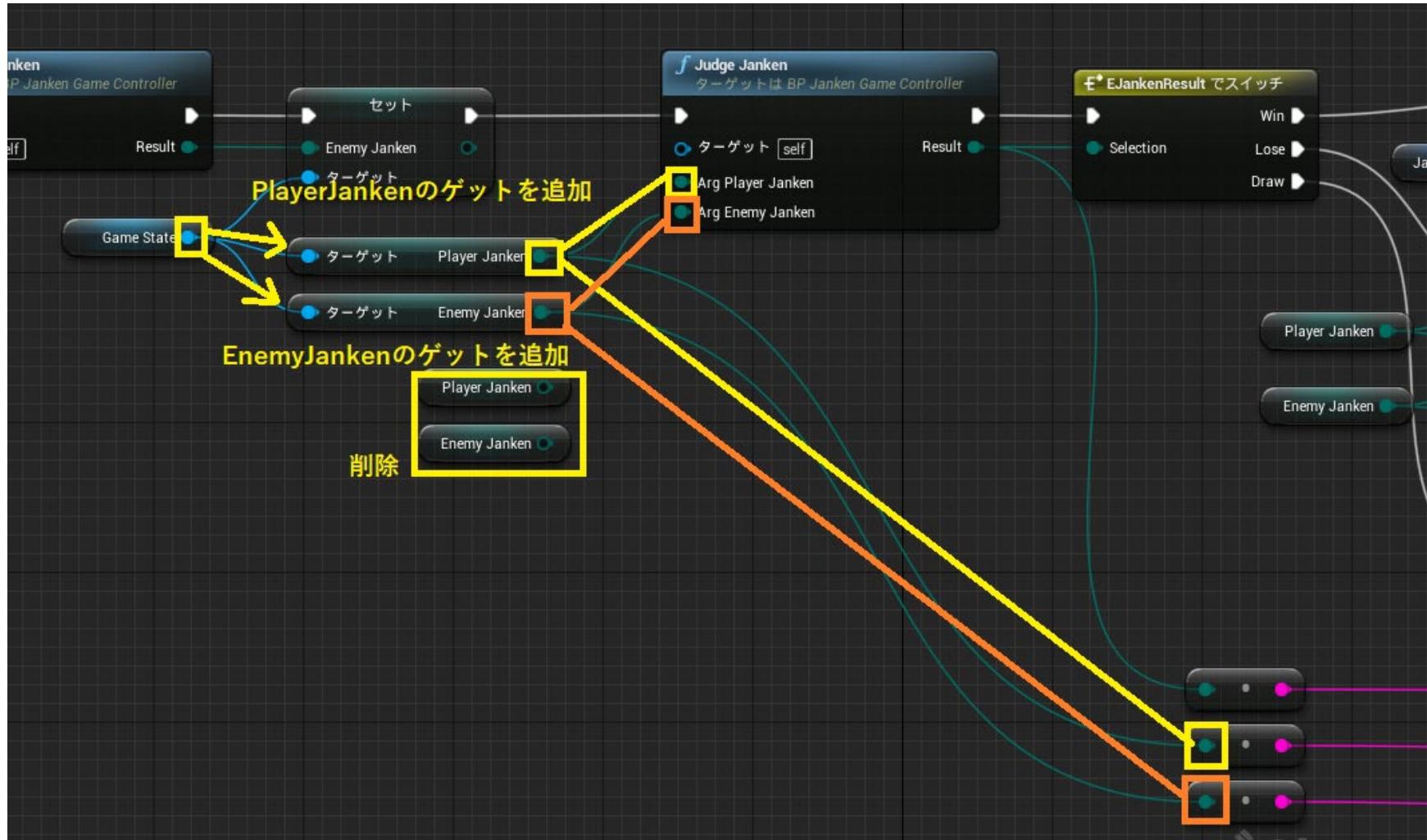


ブルー

BP_JankenGameControllerの処理をBP_GameStateの 変数を使用するように変更する 2



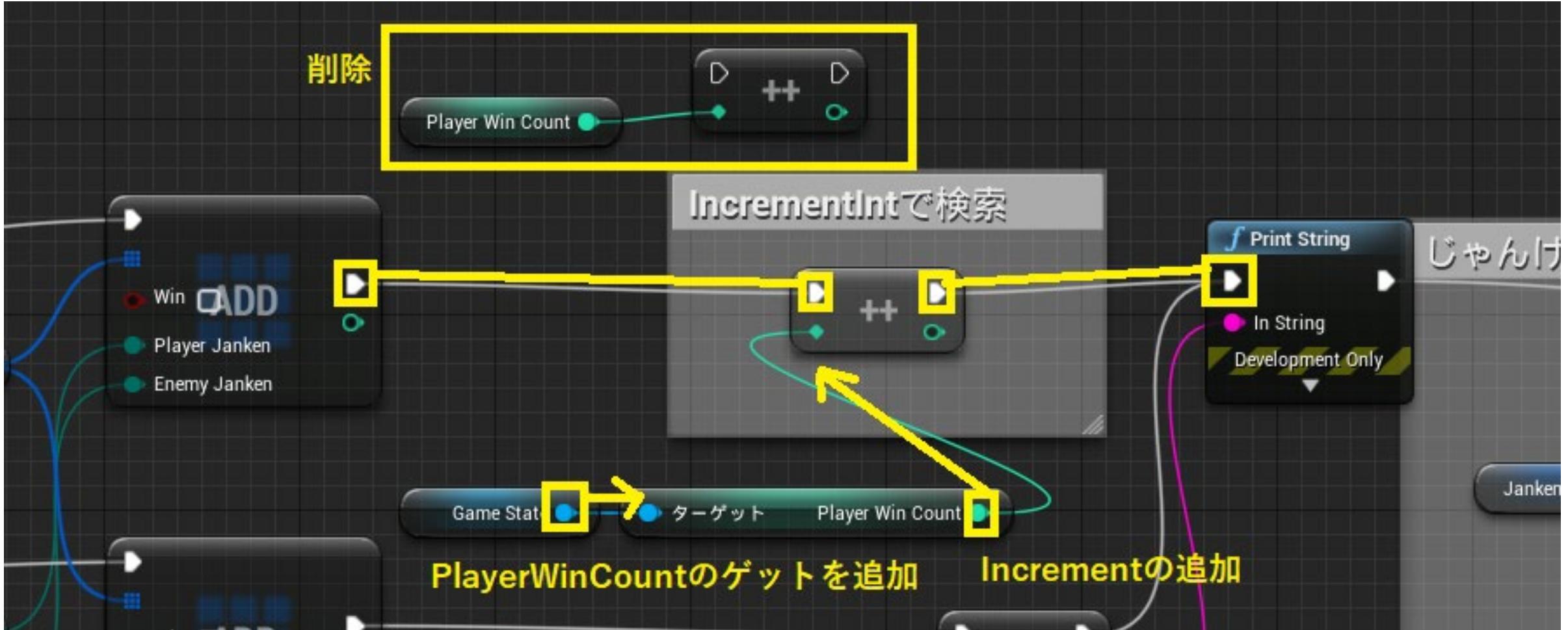
BP_JankenGameControllerの処理をBP_GameStateの 変数を使用するように変更する 3



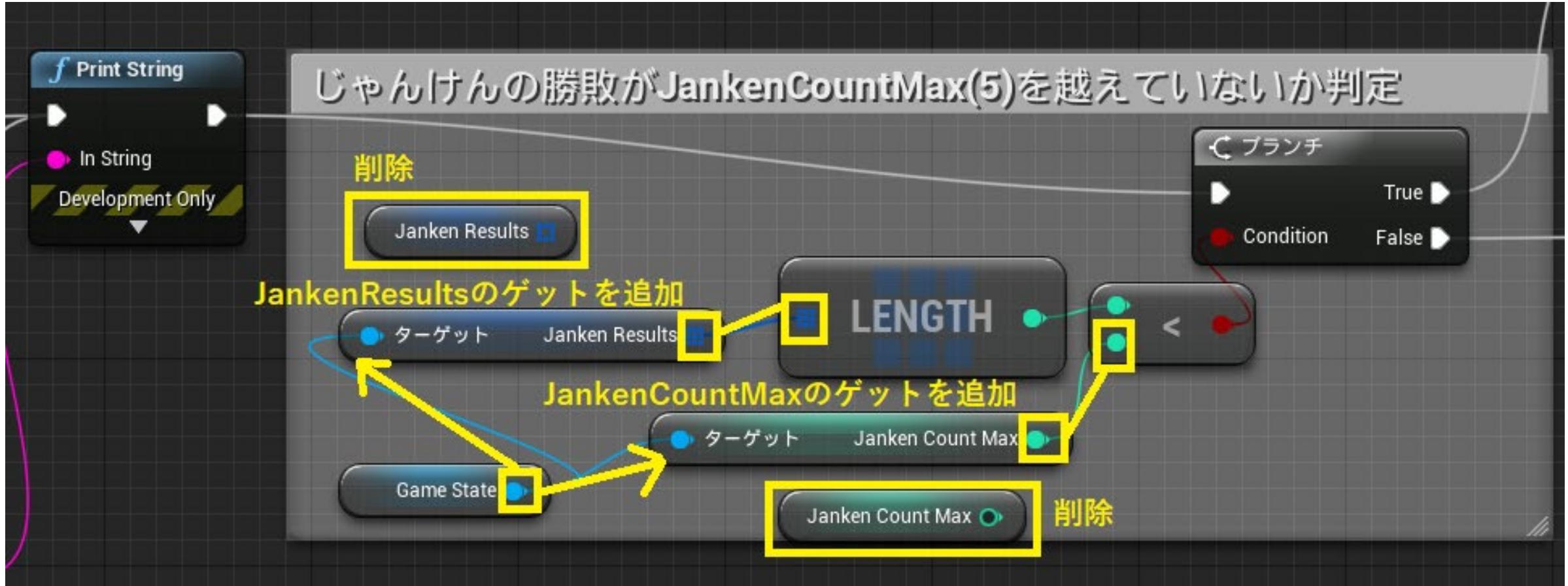
BP_JankenGameControllerの処理をBP_GameStateの 変数を使用するように変更する 4



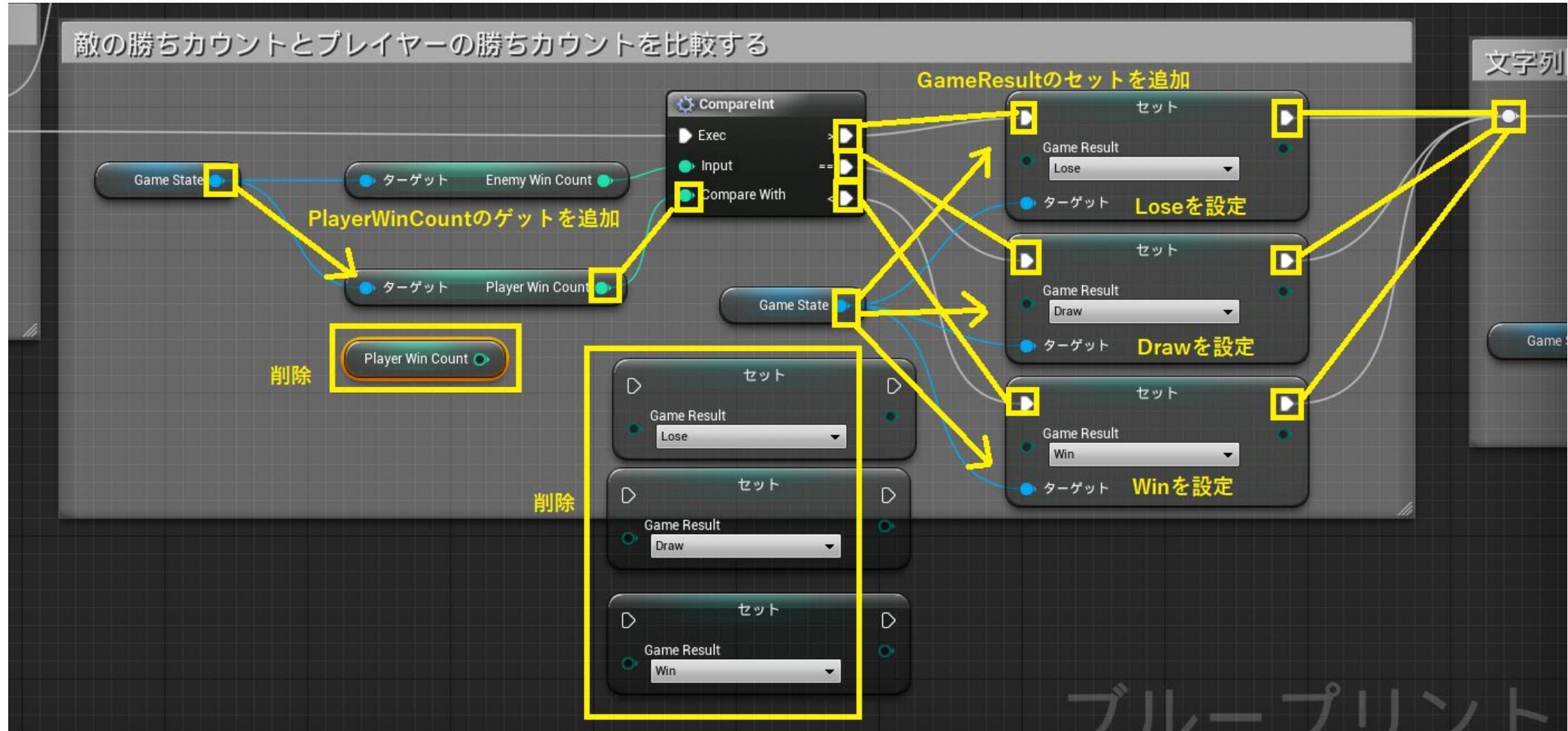
BP_JankenGameControllerの処理をBP_GameStateの 変数を使用するように変更する 5



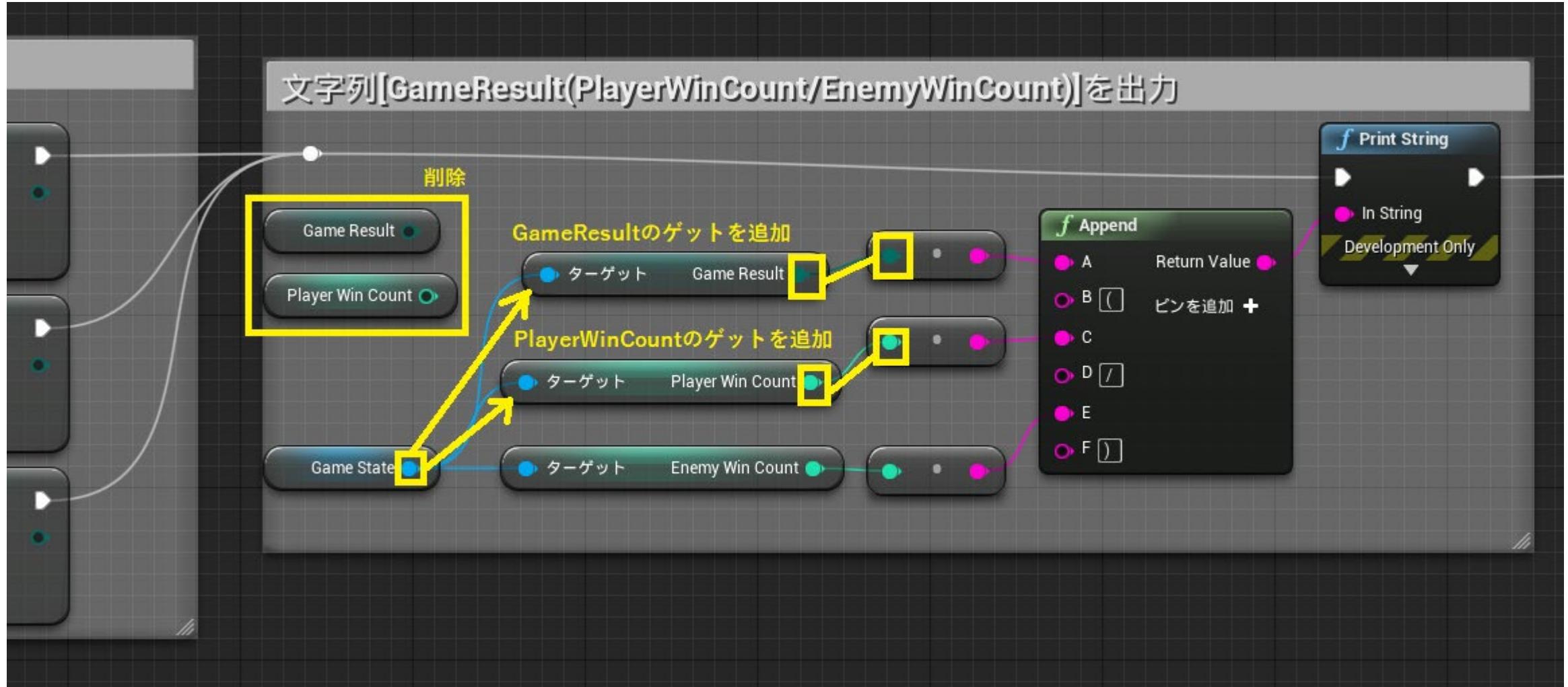
BP_JankenGameControllerの処理をBP_GameStateの
変数を使用するように変更する 6



BP_JankenGameControllerの処理をBP_GameStateの 変数を使用するように変更する 7



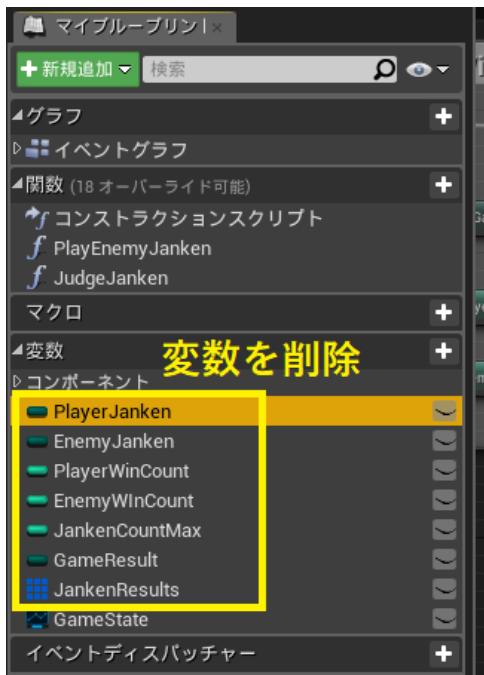
BP_JankenGameControllerの処理をBP_GameStateの 変数を使用するように変更する 8



BP_JankenGameControllerの処理をBP_GameStateの
変数を使用するように変更する 9



BP_JankenGameControllerの処理をBP_GameStateの 変数を使用するように変更する 10 BP_JankenGameControllerの変数を削除する



複数選択できないので、
1つずつ削除
GameStateは削除しない

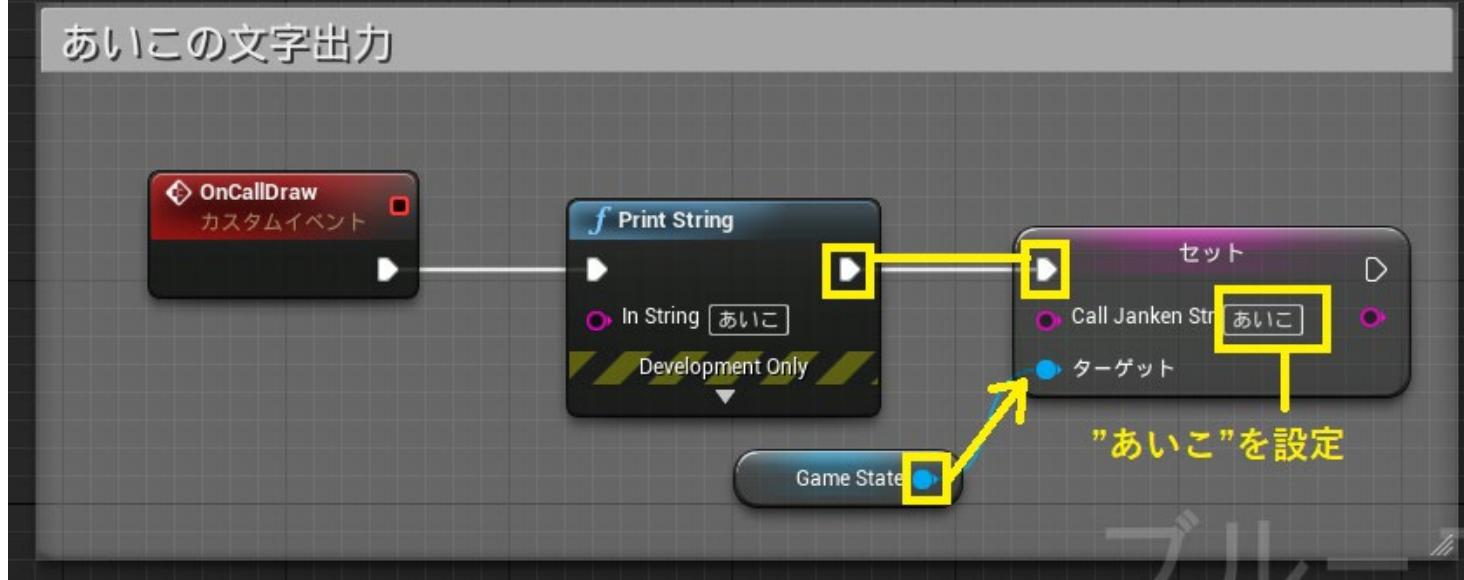
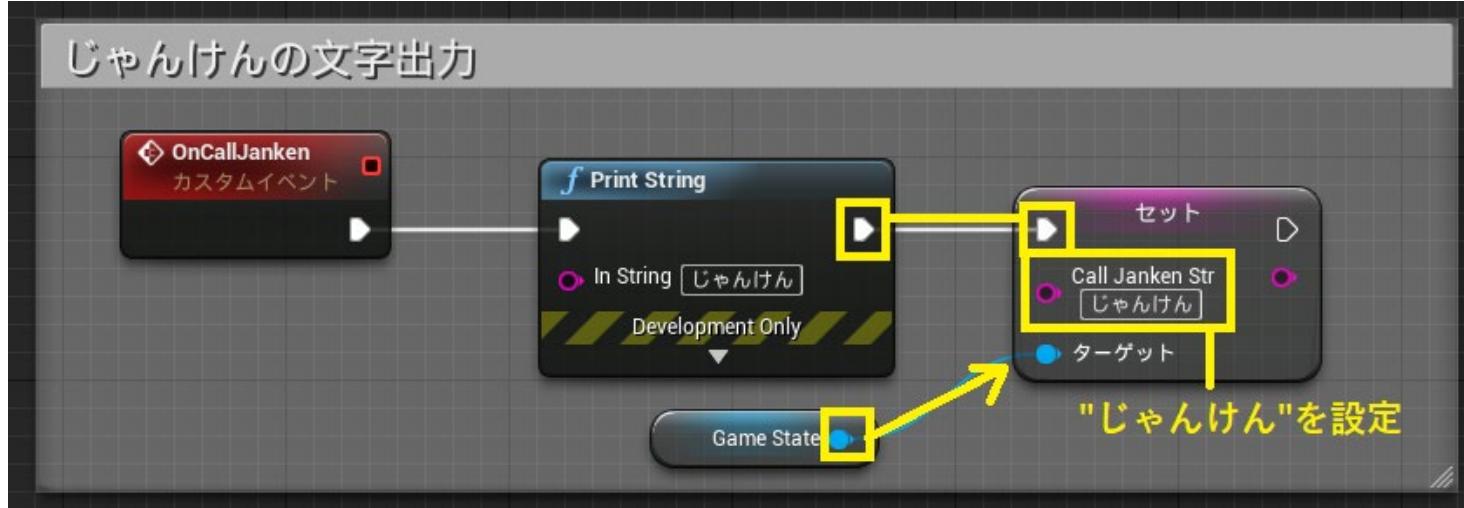


コンパイルして
エラーが出来れば完了

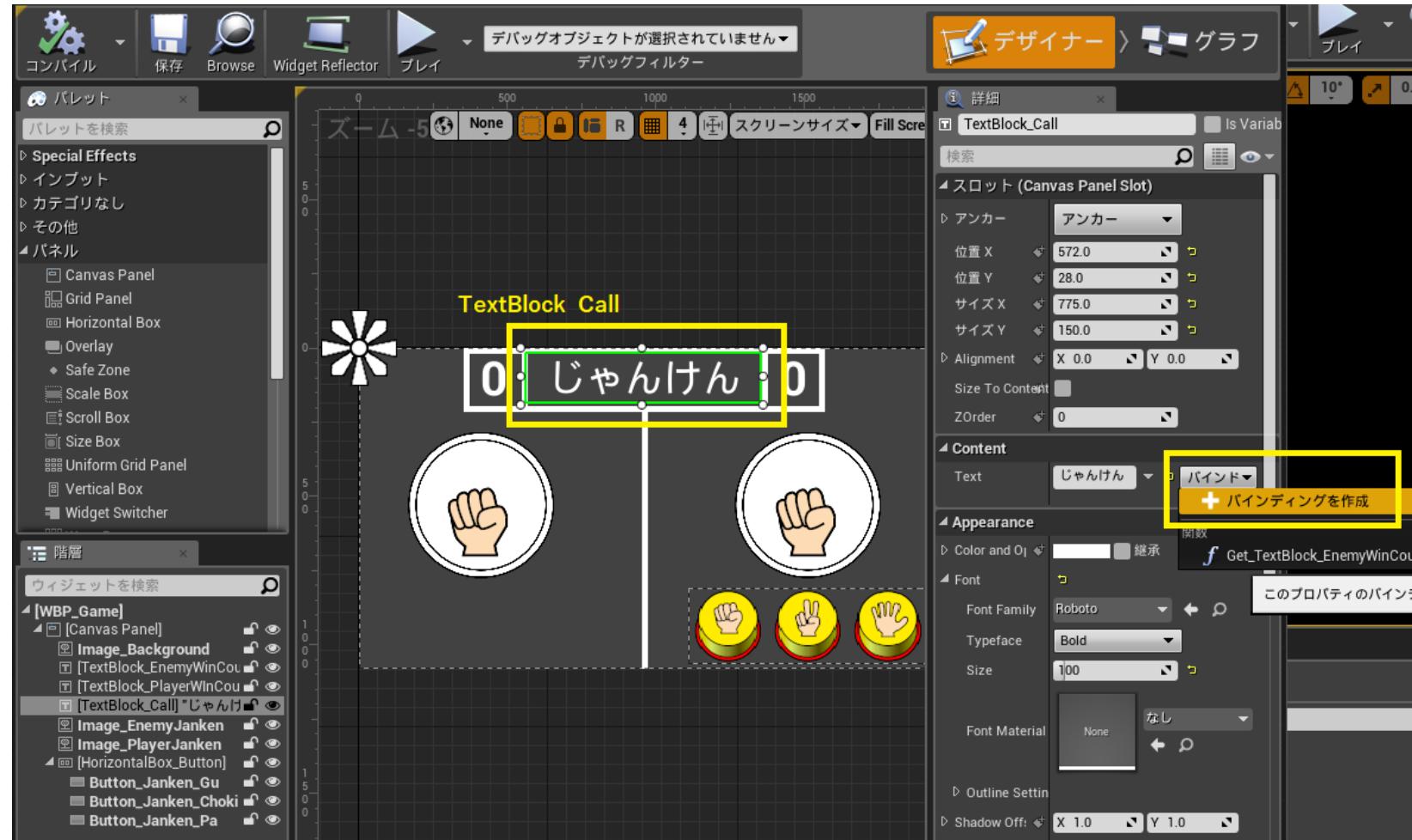
8. GameMode, GameStateを作成して、ゲームの状態を中継する

- 8.1 ウィジェットブループリントとゲームロジックの状態を中継する仕組みを作成する
- 8.2 BP_GameStateを作成する
- 8.3 BP_GameModeを作成する
- 8.4 WBP_Gameのウィジェットにバインドを作成して、BP_GameStateの値を反映させる
- 8.5 WBP_Gameを表示する
- 8.6 ゲームの状態を中継する仕組みの説明
- 8.7 BP_JankenGameControllerの変数をBP_GameStateに移行する
- 8.8 BP_GameStateに関数Initializeを作成し、初期化処理を置き換える
- 8.9 BP_JankenGameControllerの処理をBP_GameStateの変数を使用するように変更する
- 8.10 BP_GameStateのCallJankenStrに文字を設定する
- 8.11 BP_GameStateの変数をWBP_Gameのウィジェットにバインドする

BP_GameStateのCallJankenStrに文字を設定する



TextBlock_Callのテキストにバインディングを作成する 1



TextBlock_Callのテキストにバインディングを作成する 2



8. GameMode, GameStateを作成して、ゲームの状態を中継する

- 8.1 ウィジェットブループリントとゲームロジックの状態を中継する仕組みを作成する
- 8.2 BP_GameStateを作成する
- 8.3 BP_GameModeを作成する
- 8.4 WBP_Gameのウィジェットにバインドを作成して、BP_GameStateの値を反映させる
- 8.5 WBP_Gameを表示する
- 8.6 ゲームの状態を中継する仕組みの説明
- 8.7 BP_JankenGameControllerの変数をBP_GameStateに移行する
- 8.8 BP_GameStateに関数Initializeを作成し、初期化処理を置き換える
- 8.9 BP_JankenGameControllerの処理をBP_GameStateの変数を使用するように変更する
- 8.10 BP_GameStateのCallJankenStrに文字を設定する
- 8.11 BP_GameStateの変数をWBP_Gameのウィジェットにバインドする

純粹キャストに変更



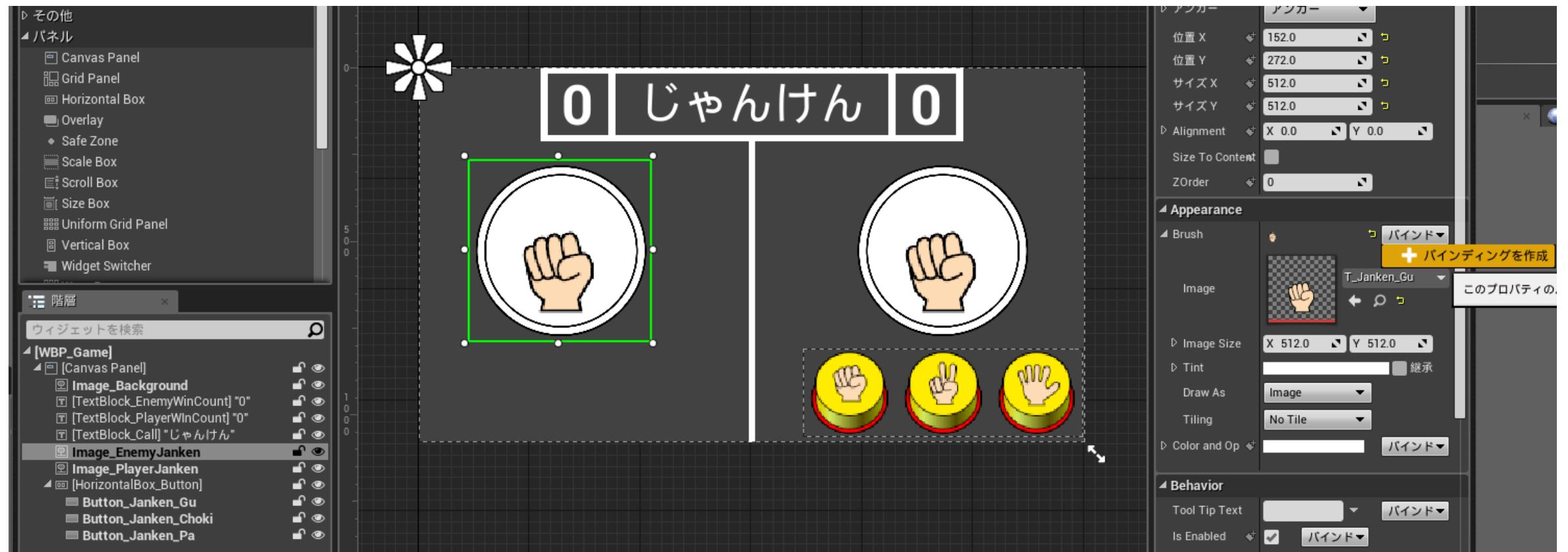
TextBlock_PlayerWinCountにバインディングを作成する 1



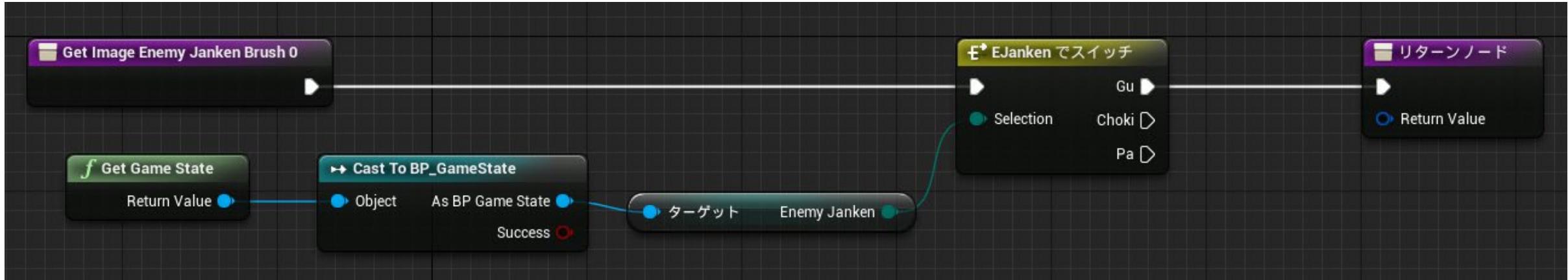
TextBlock_PlayerWinCountにバインディングを作成する 2



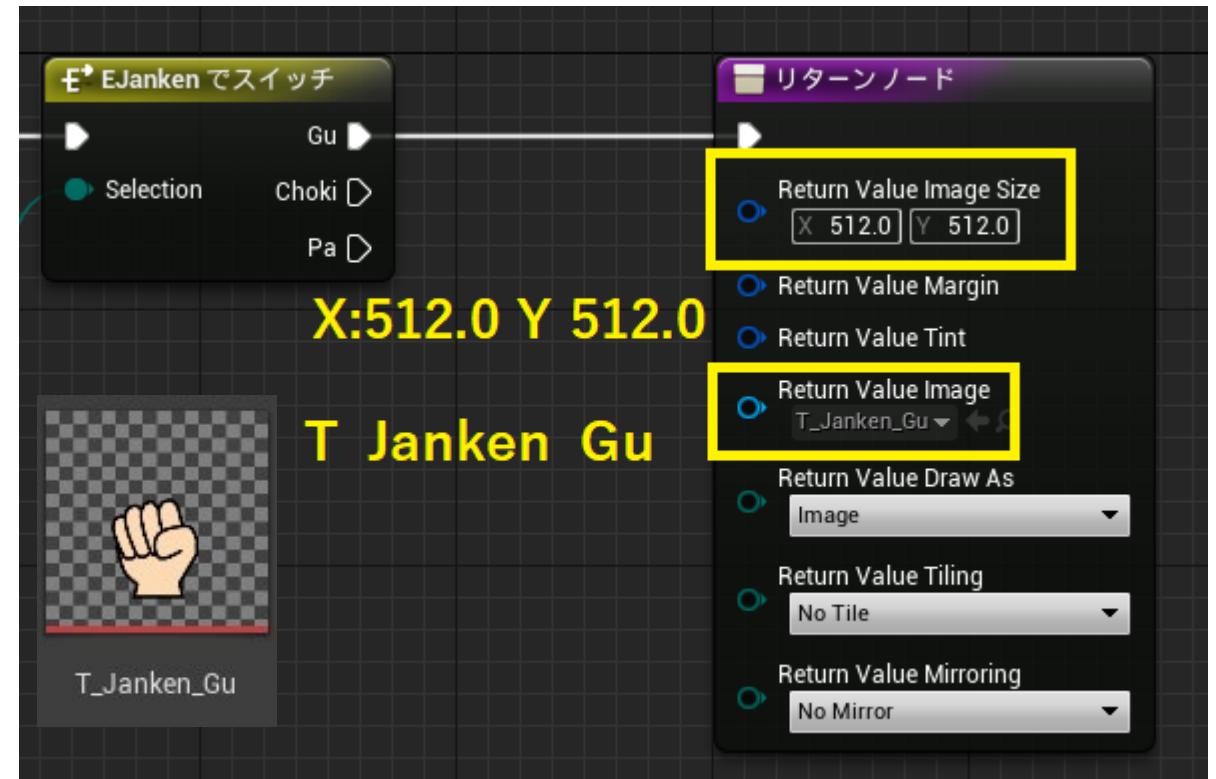
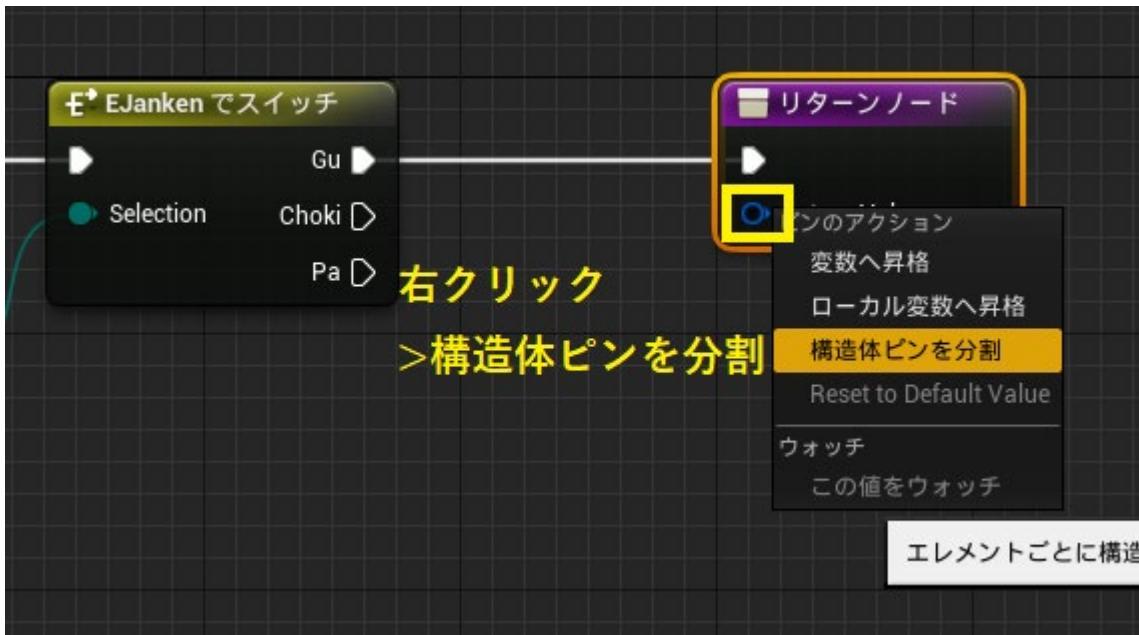
Image_Enemy_JankenのBrushにバインディングを作成する 1



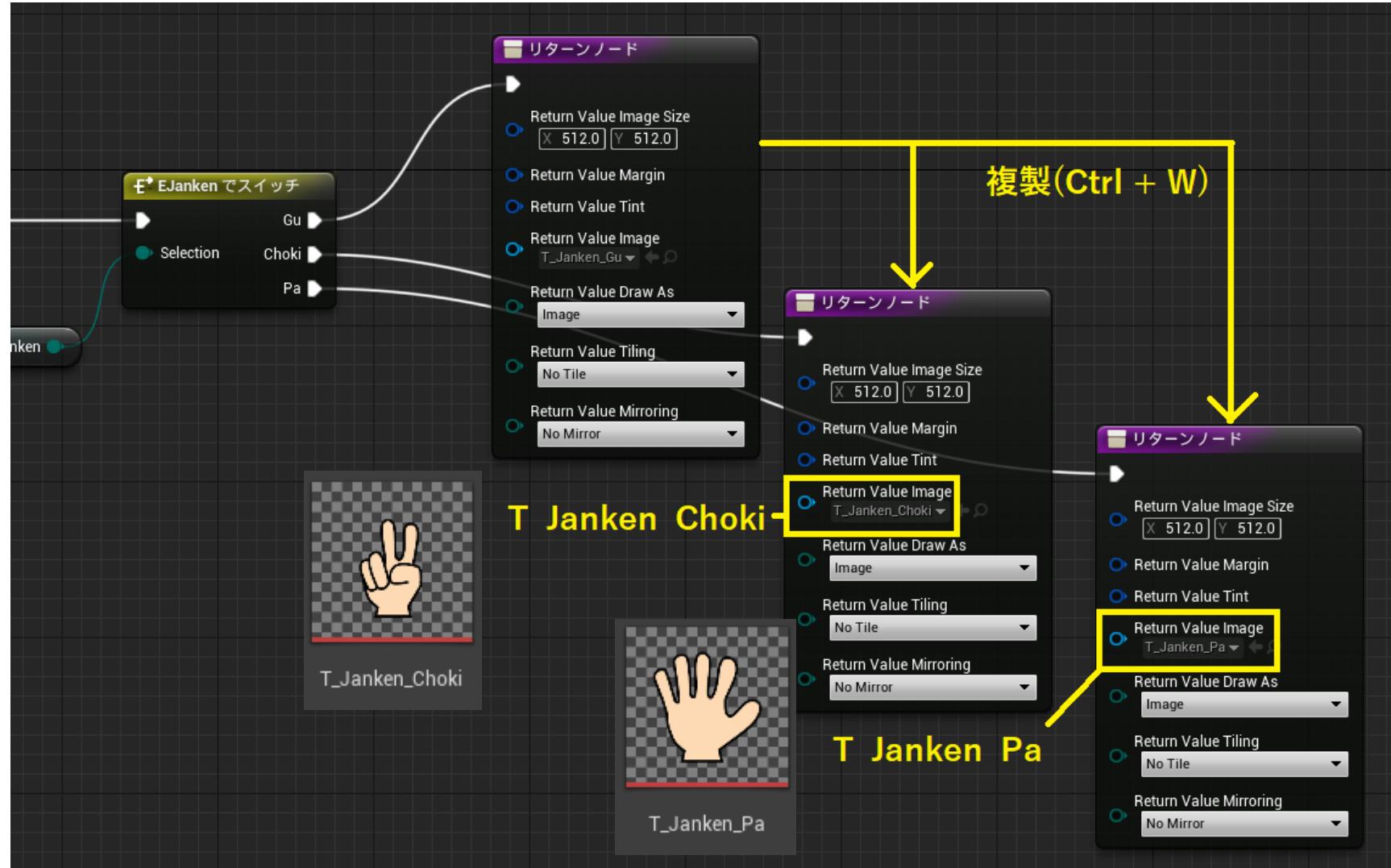
Image_Enemy_JankenのBrushにバインディングを作成する 2



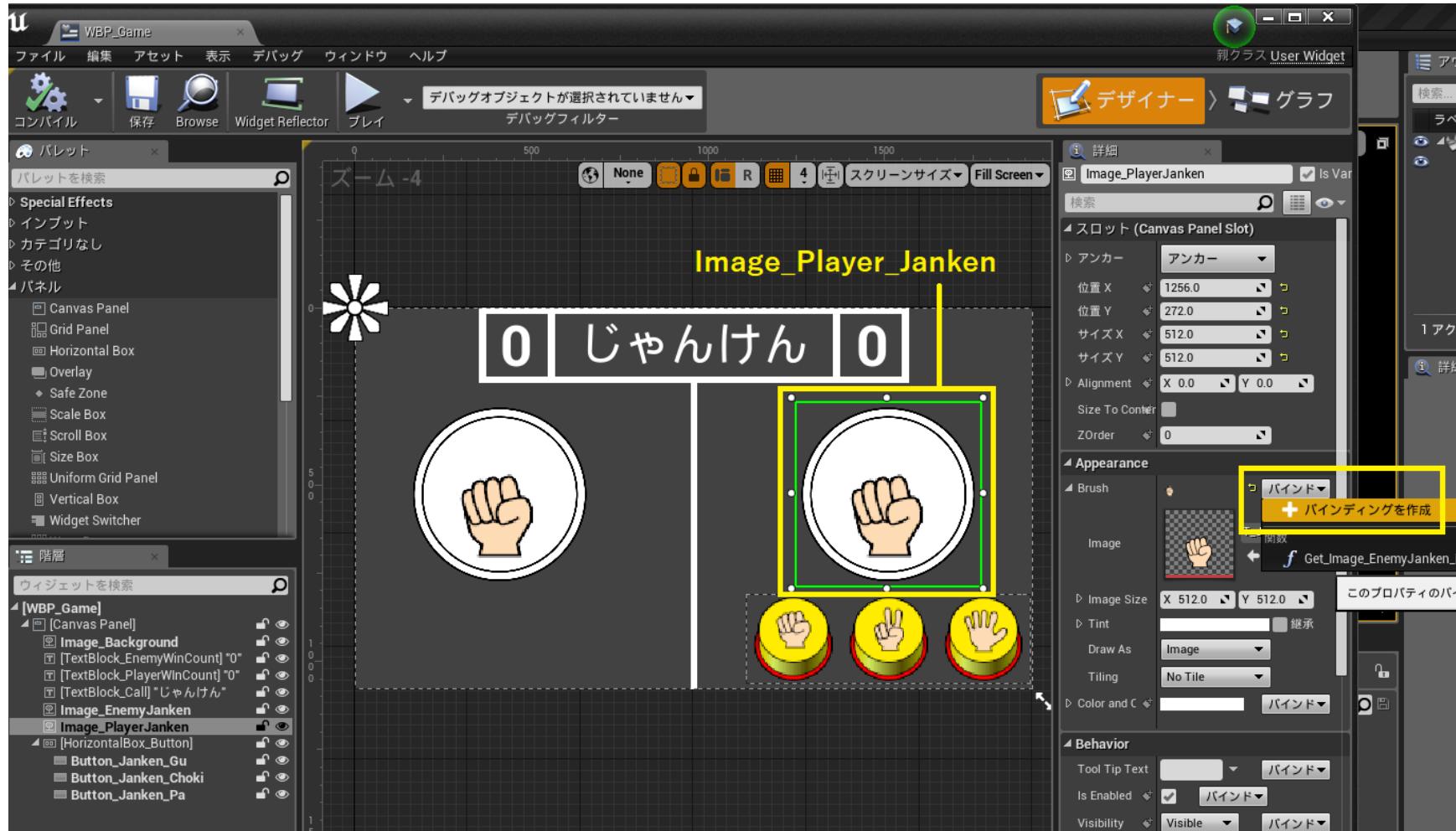
Image_Enemy_JankenのBrushにバインディングを作成する 3



Image_Enemy_JankenのBrushにバインディングを作成する 4



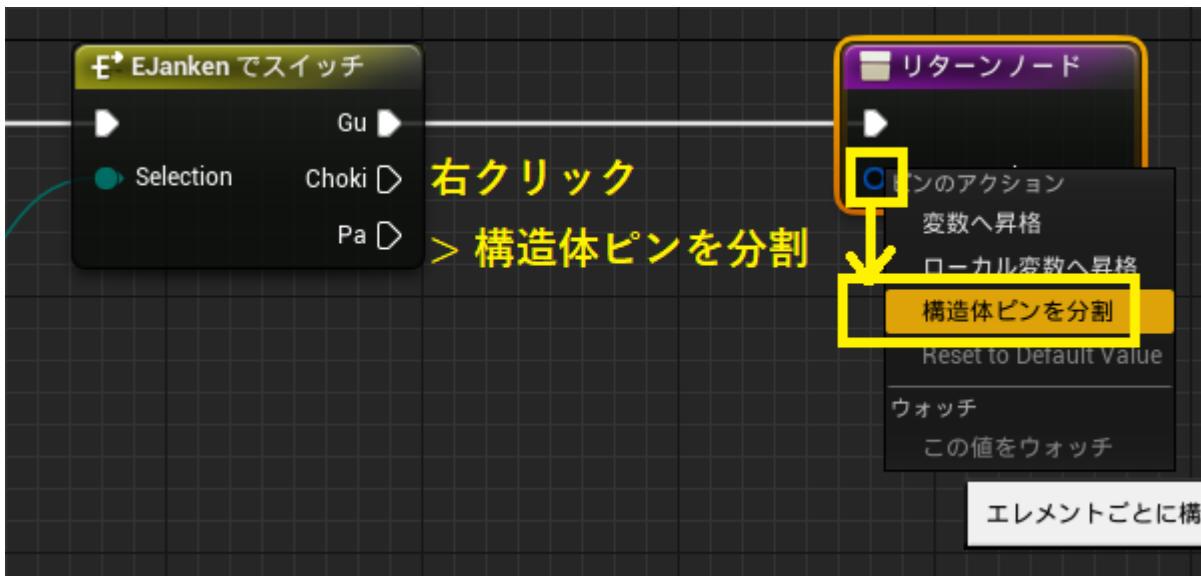
Image_Player_JankenのBrushにバインディングを作成する 1



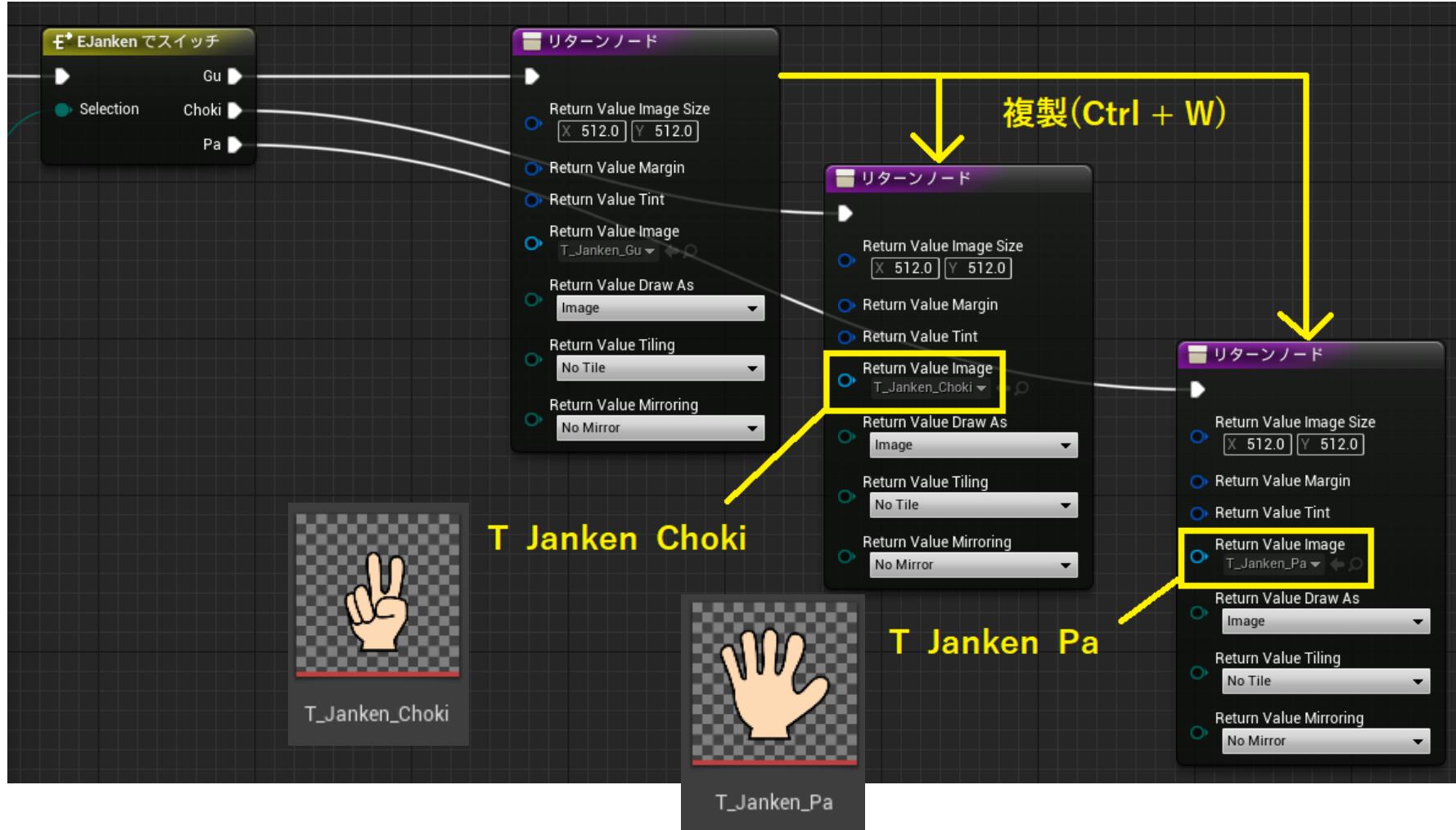
Image_Player_JankenのBrushにバインディングを作成する 2



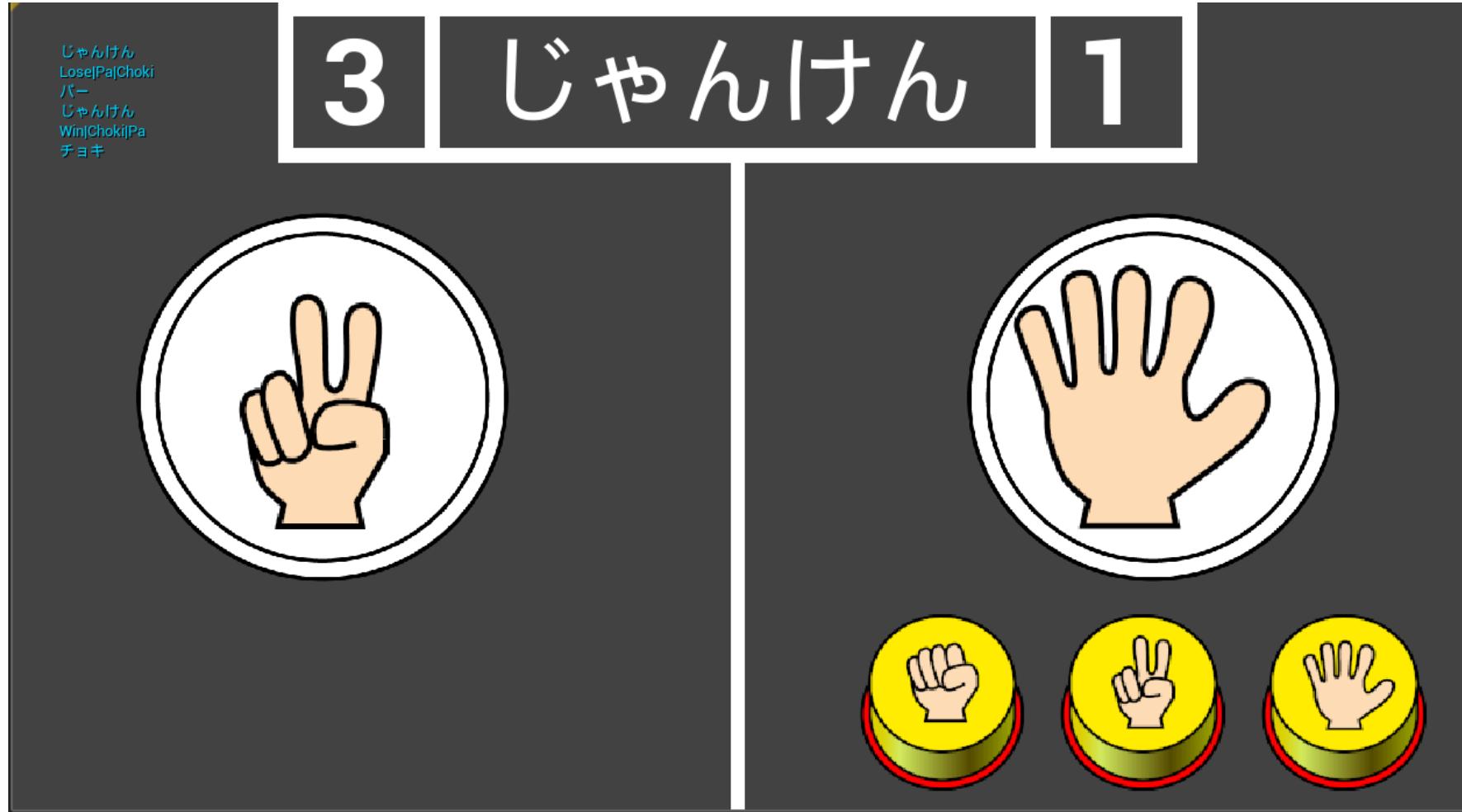
Image_Player_JankenのBrushにバインディングを作成する 3



Image_Player_JankenのBrushにバインディングを作成する 4



プレイして確認



9. ウィジェットブループリントのイベントをブループリントで受け取って処理する

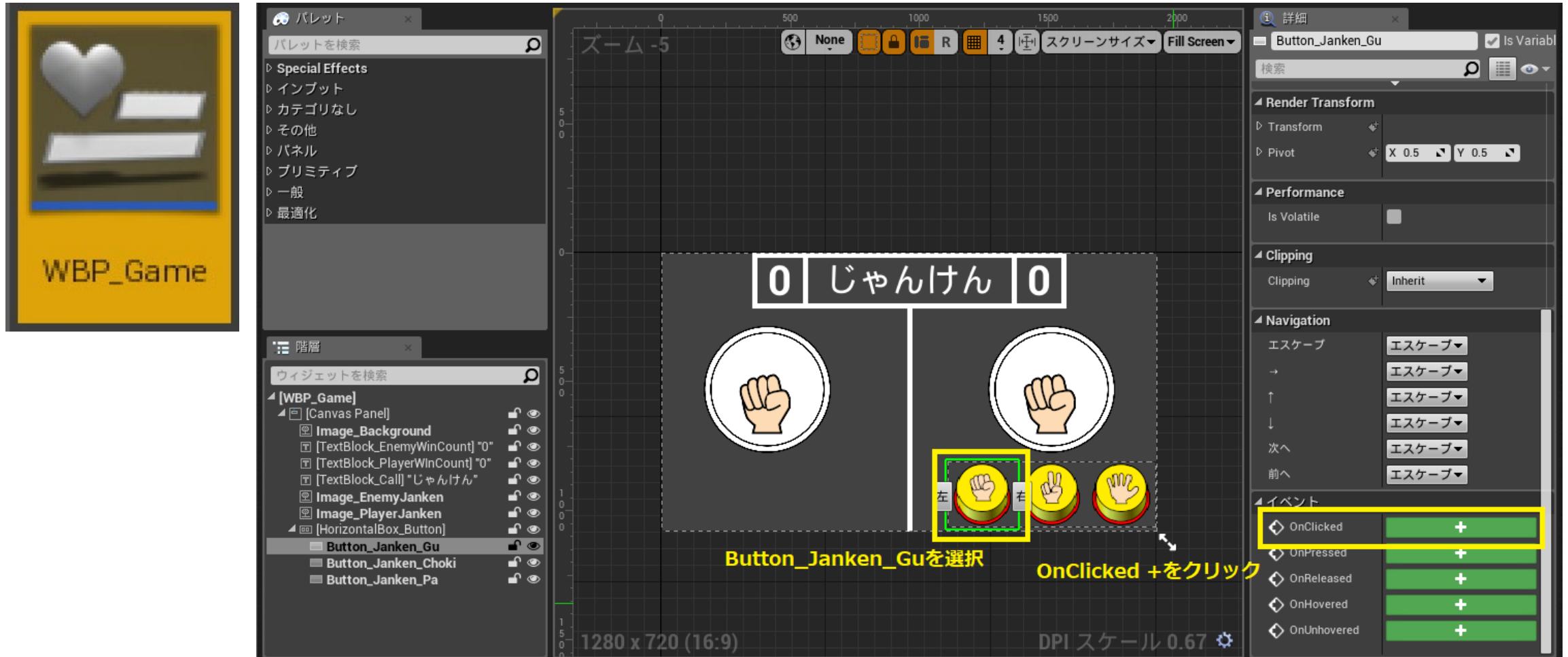
9. ウィジェットブループリントのイベントをブループリントで受け取って処理する

- 9.1 WBP_GameのButton_Janken_Guをクリックしたら、グーを選択した処理を実行するように実装する
- 9.2 Button_Janken_Choki,Button_Janken_Paも同様にクリックしたら、処理を実行するように実装する
- 9.3 MouseOverした際にプレイヤーのじゃんけん画像が変更するように実装する

9. ウィジェットブループリントのイベントをブループリントで受け取って処理する

- 9.1 WBP_GameのButton_Janken_Guをクリックしたら、グーを選択した処理を実行するように実装する
- 9.2 Button_Janken_Choki,Button_Janken_Paも同様にクリックしたら、処理を実行するように実装する
- 9.3 MouseOverした際にプレイヤーのじゃんけん画像が変更するように実装する

Button_Janken_GuにOnClickedイベントを追加する 1

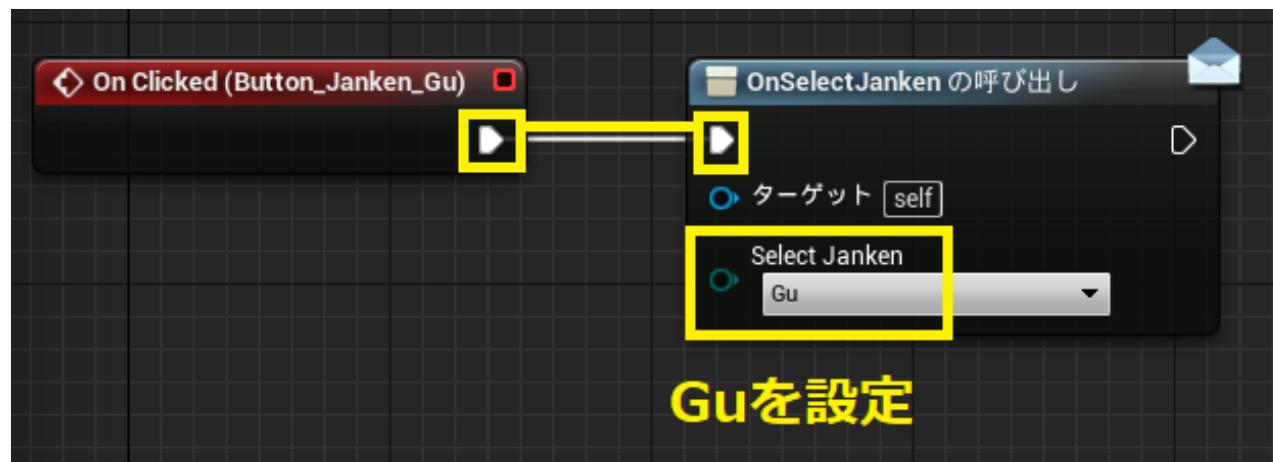
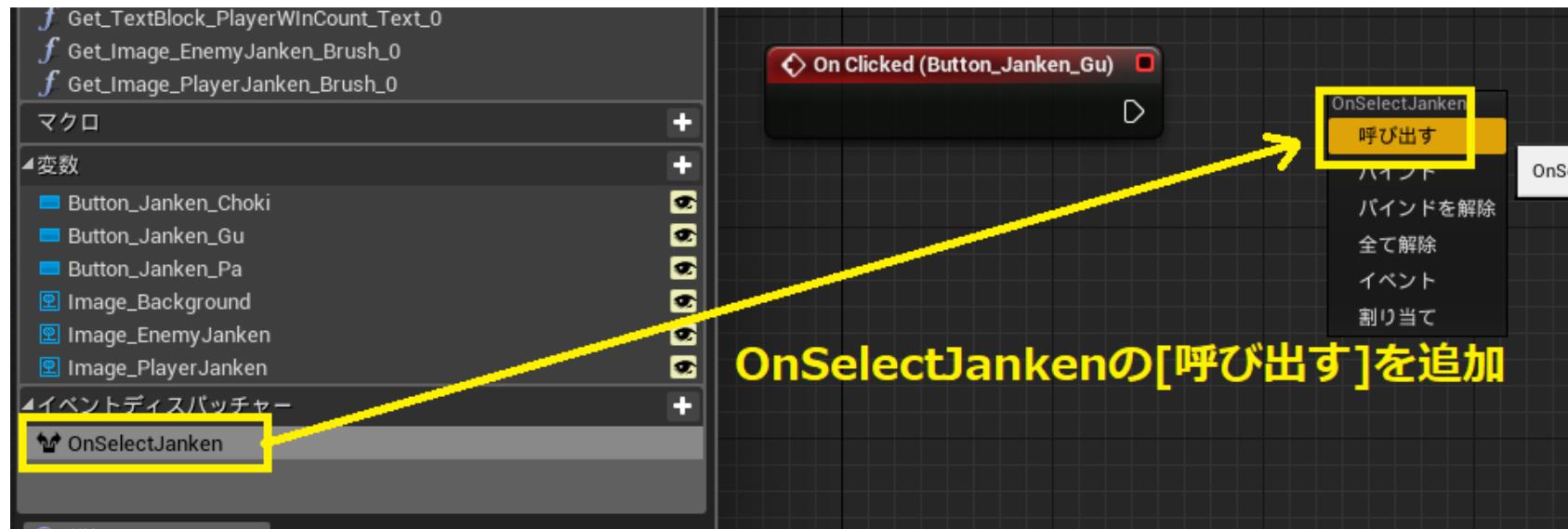


Button_Janken_GuにOnClickedイベントを追加する 2

The image shows two screenshots of the Unity Editor interface. The left screenshot displays the Inspector window for a game object, showing variables like Button_Janken_Choki, Button_Janken_Gu, etc., and an event named OnSelectJanken. A yellow box highlights the 'OnSelectJanken' event. Below it, a note says '名前をOnSelectJankenに設定' (Set the name to OnSelectJanken). The right screenshot shows the Event Graph window, where a new event 'OnSelectJanken' is being created. A yellow box highlights the '+ イベントディスパッチャー' button. The input configuration section is shown with a yellow box around the 'SelectJanken' input and a note 'インプットを設定' (Set Input). A table below shows the parameter mapping:

パラメータ名	型
SelectJanken	EJanken

Button_Janken_GuにOnClickedイベントを追加する 3



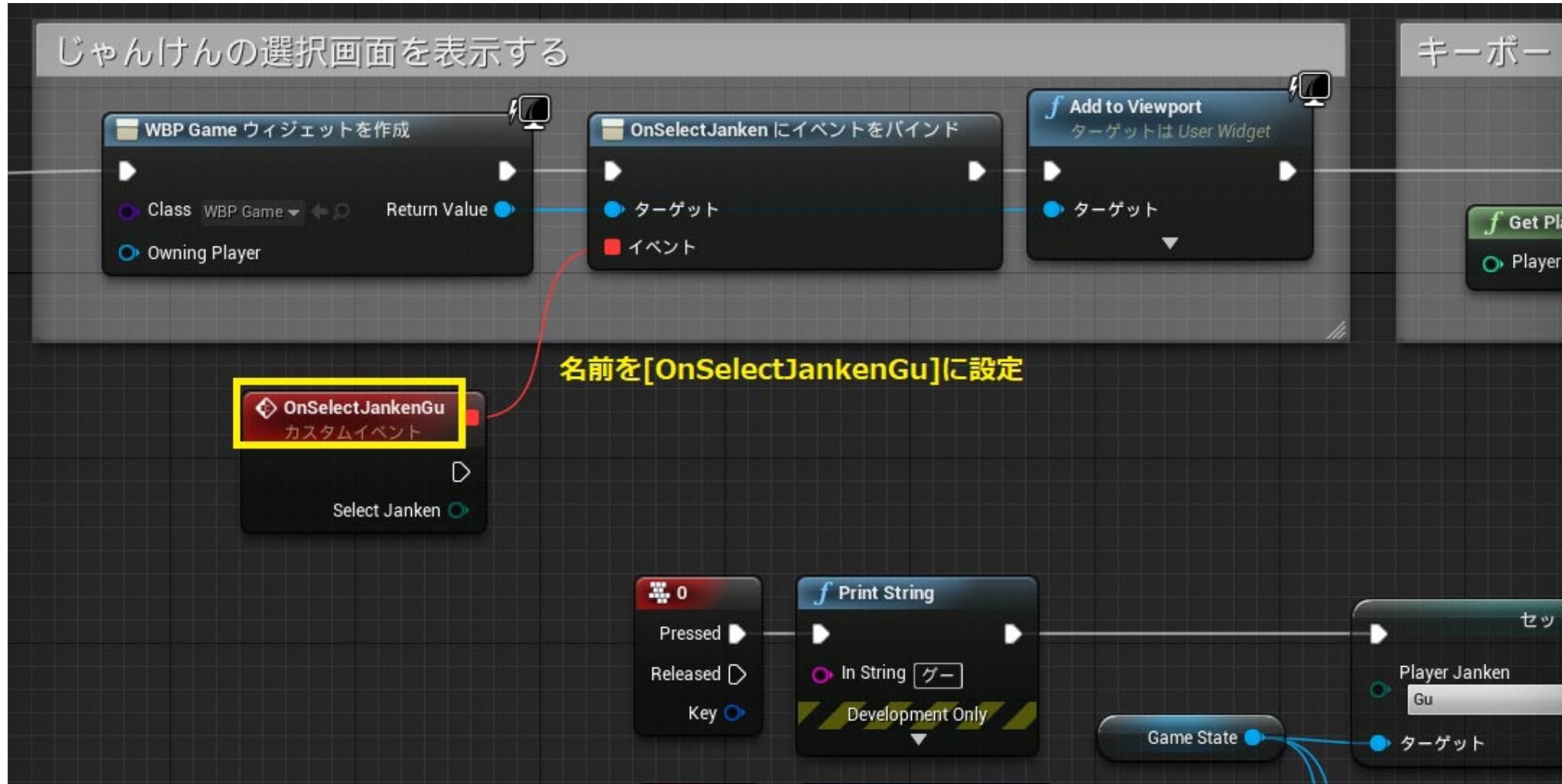
OnSelectJankenにイベントをバインド 1



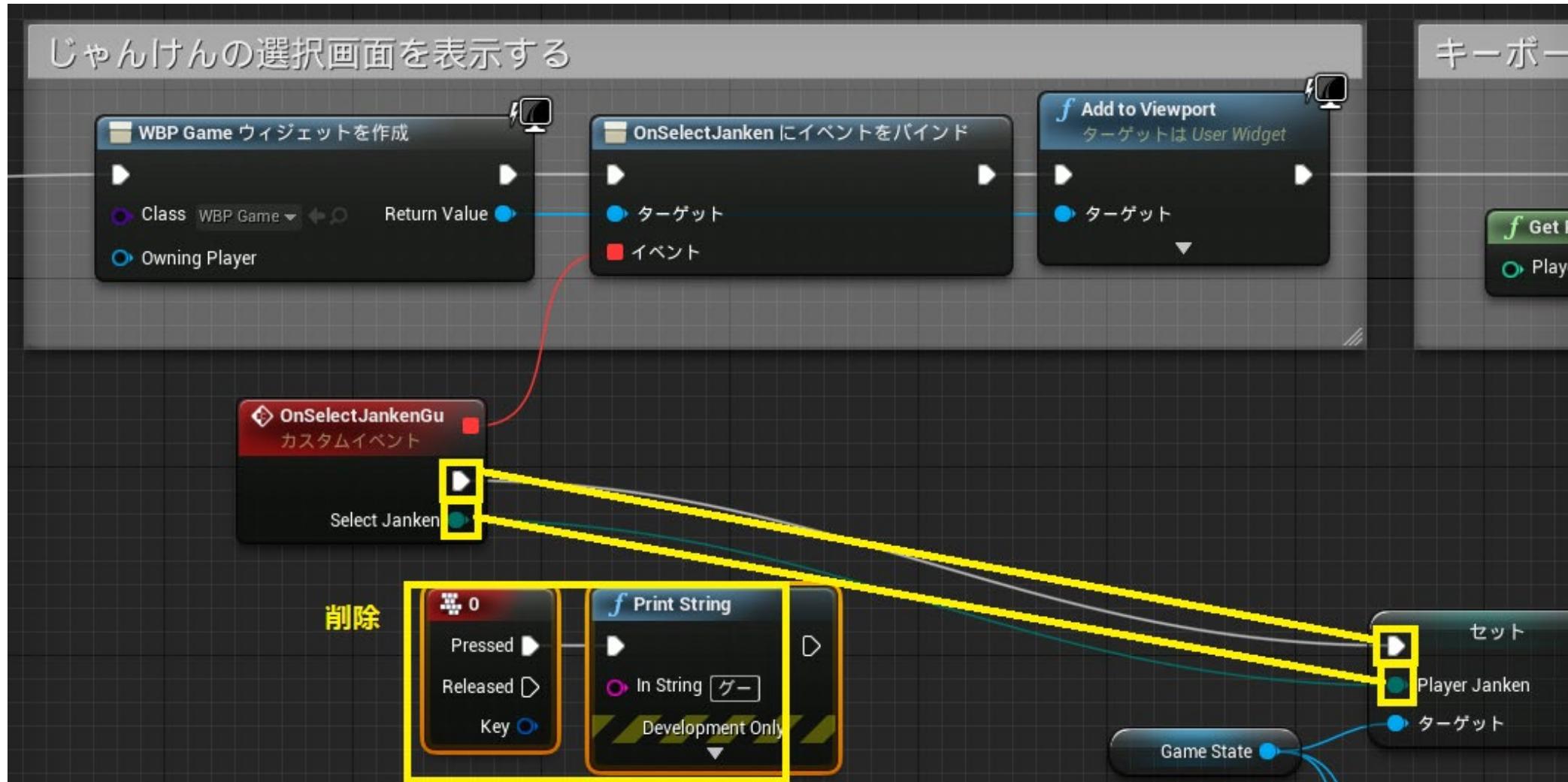
OnSelectJankenにイベントをバインド 2



OnSelectJankenにイベントをバインド 3



OnSelectJankenにイベントを BIND 4



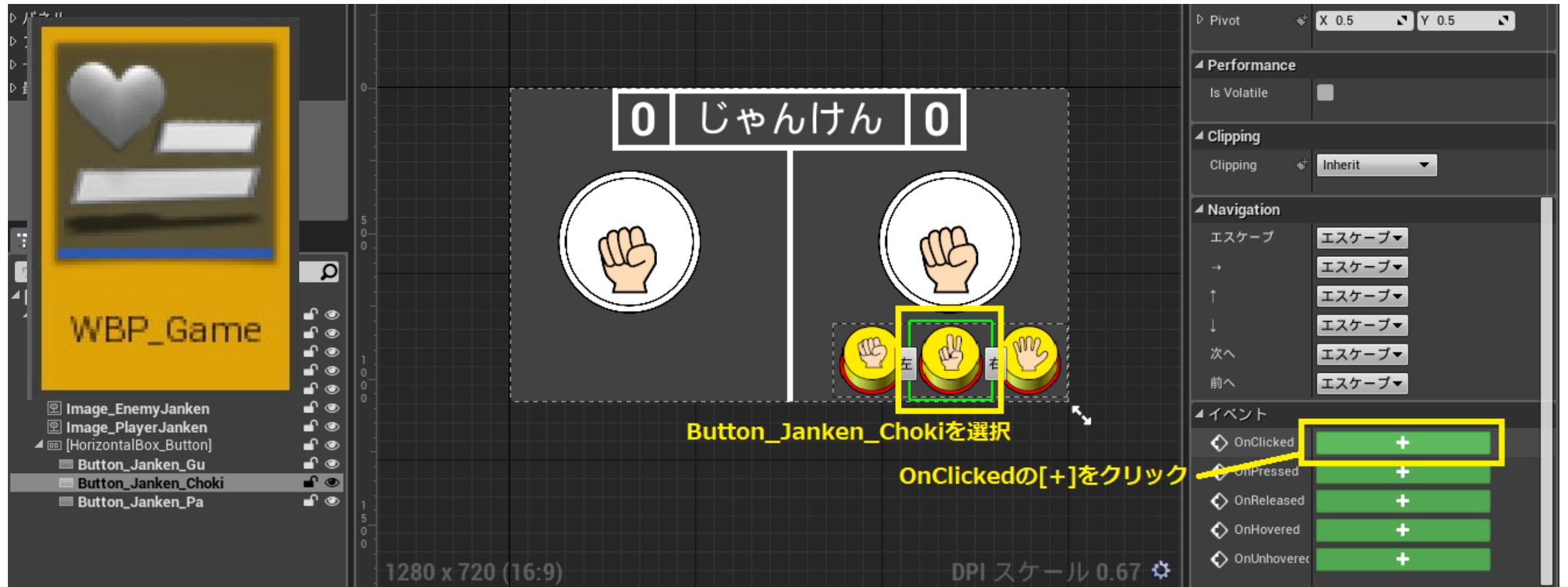
プレイして確認する



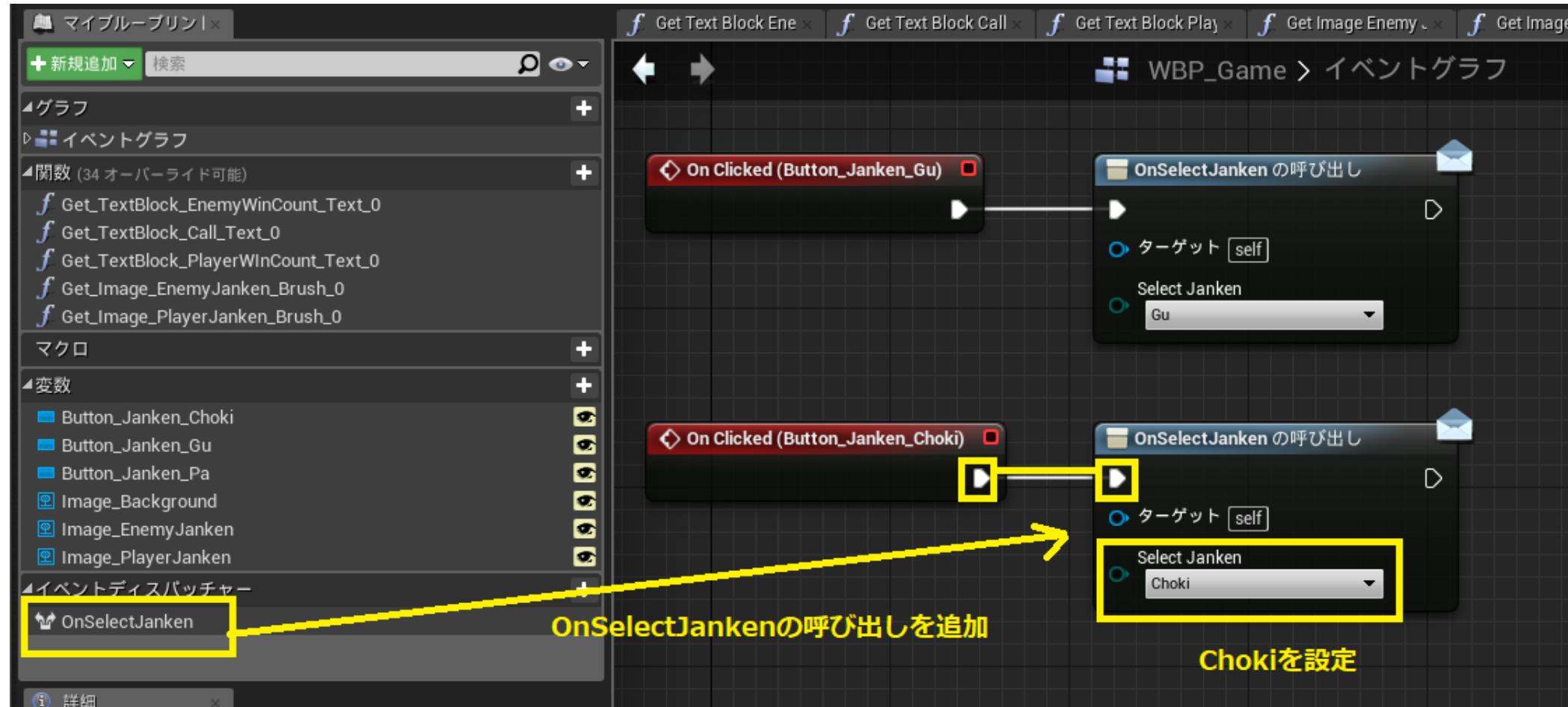
9. ウィジェットブループリントのイベントをブループリントで受け取って処理する

- 9.1 WBP_GameのButton_Janken_Guをクリックしたら、グーを選択した処理を実行するように実装する
- 9.2 Button_Janken_Choki,Button_Janken_Paも同様にクリックしたら、処理を実行するように実装する
- 9.3 MouseOverした際にプレイヤーのじゃんけん画像が変更するように実装する

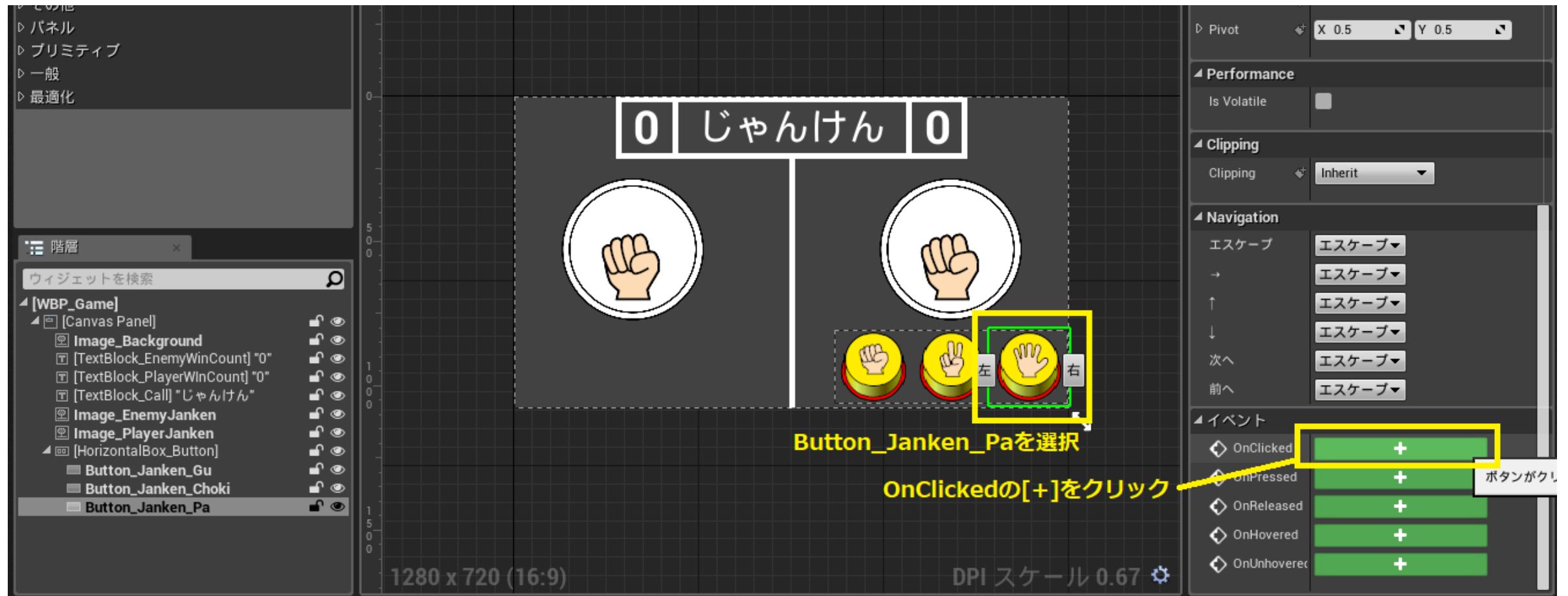
Button_Janken_ChokiにOnClickedイベントを追加する 1



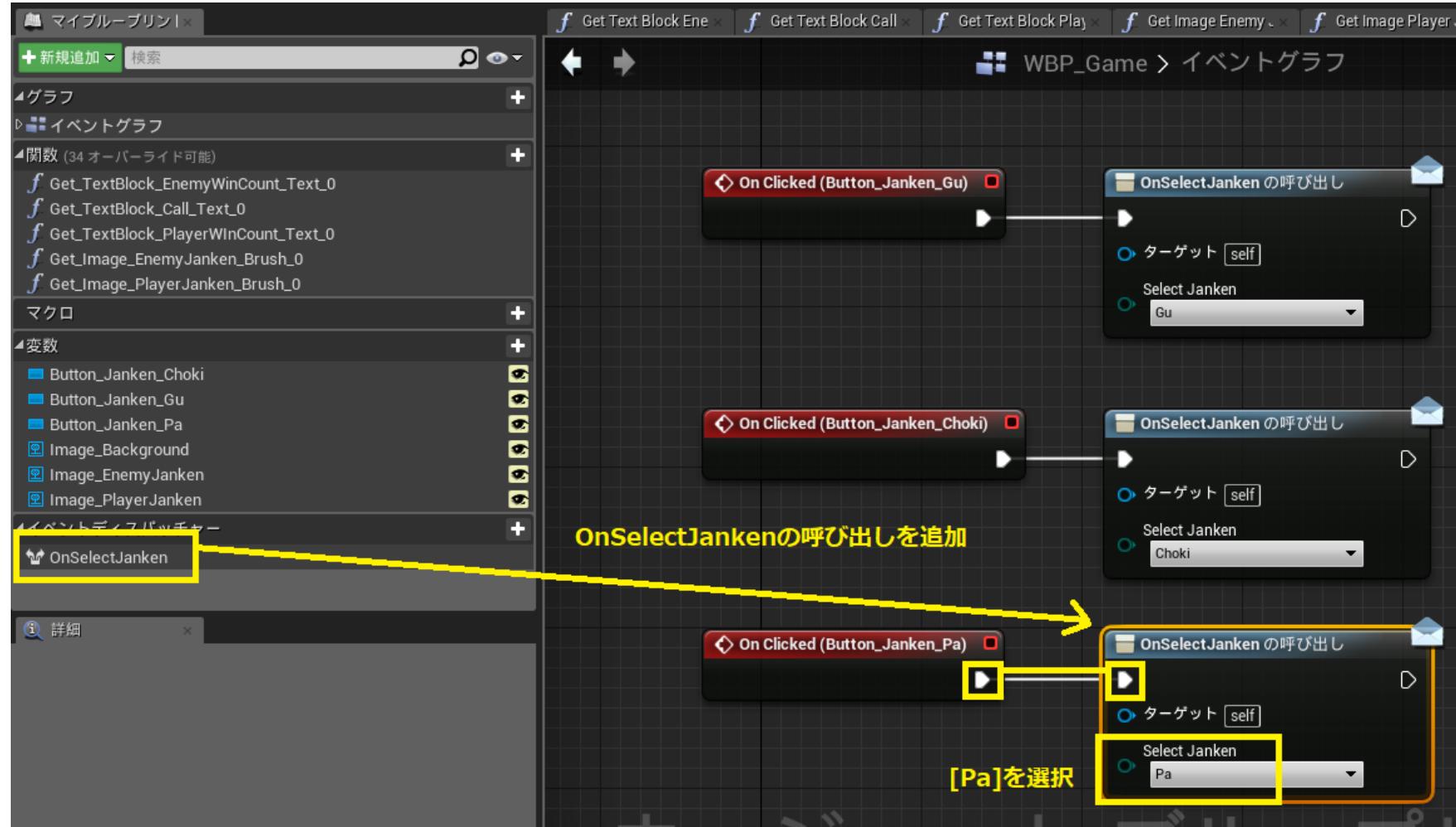
Button_Janken_ChokiにOnClickedイベントを追加する 2



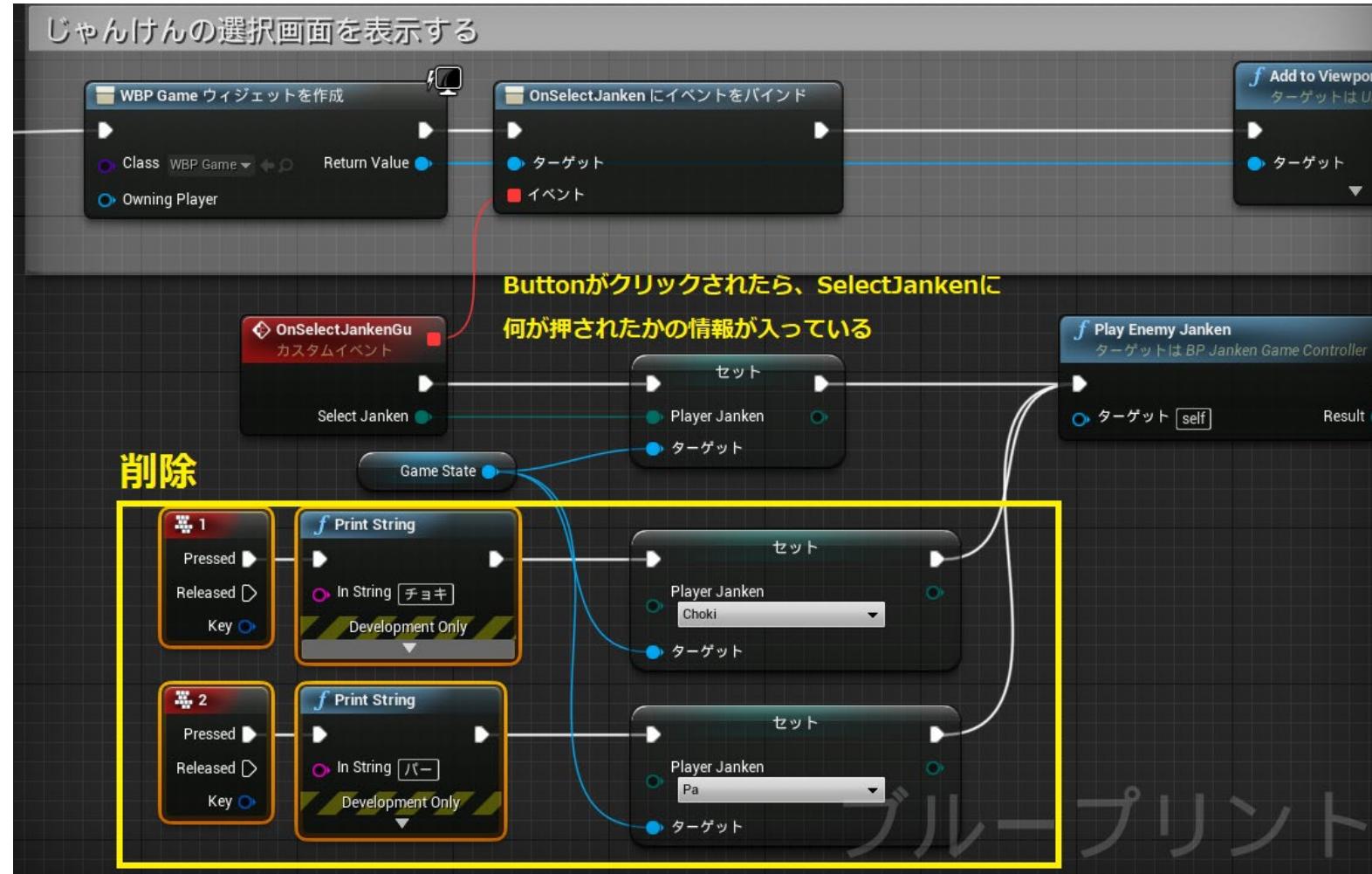
Button_Janken_PaにOnClickedイベントを追加する 1



Button_Janken_PaにOnClickedイベントを追加する 2



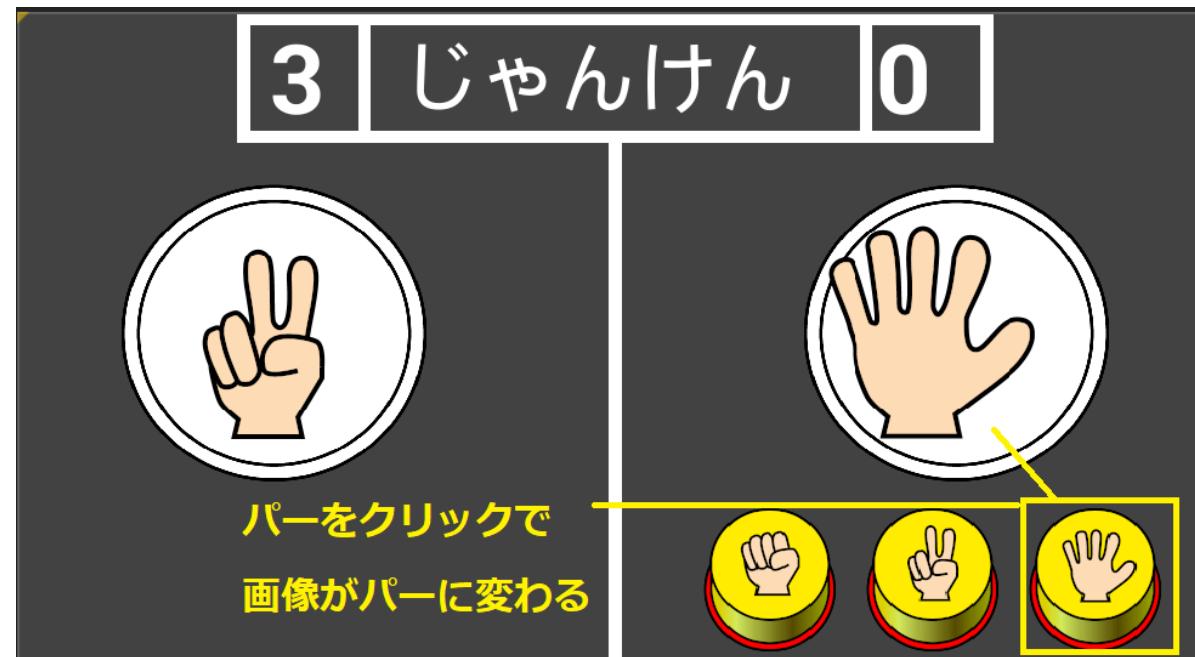
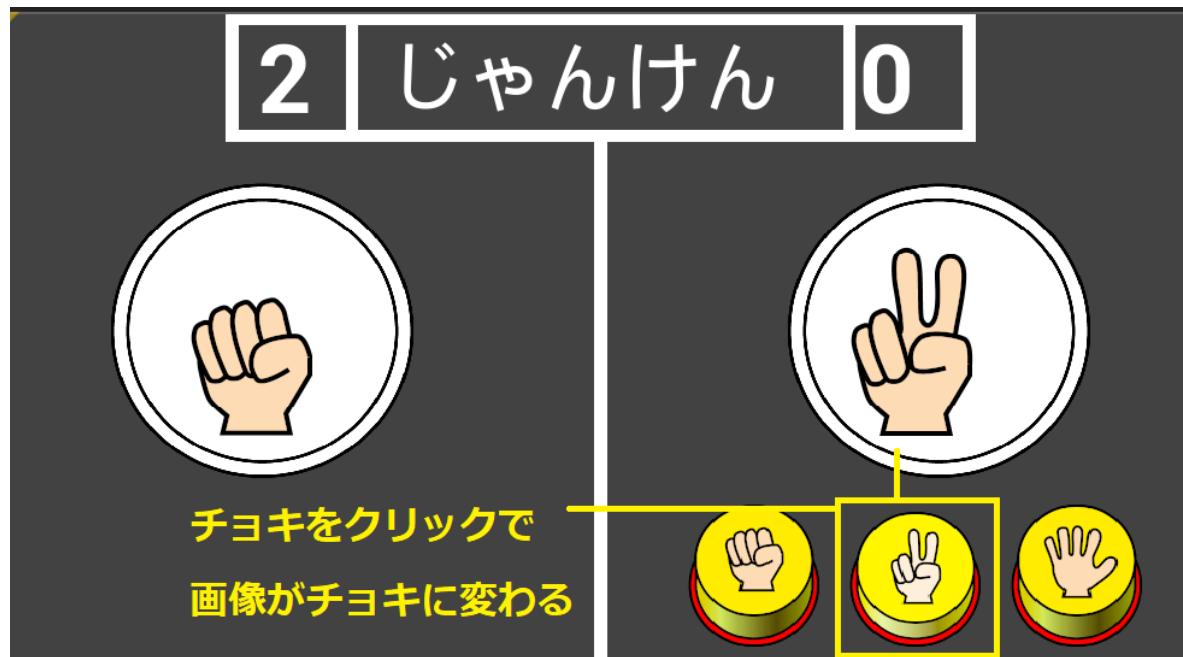
OnSelectJankenにイベントをバインド キー ボード イベント 1,2 を削除



マウスを常に表示させる



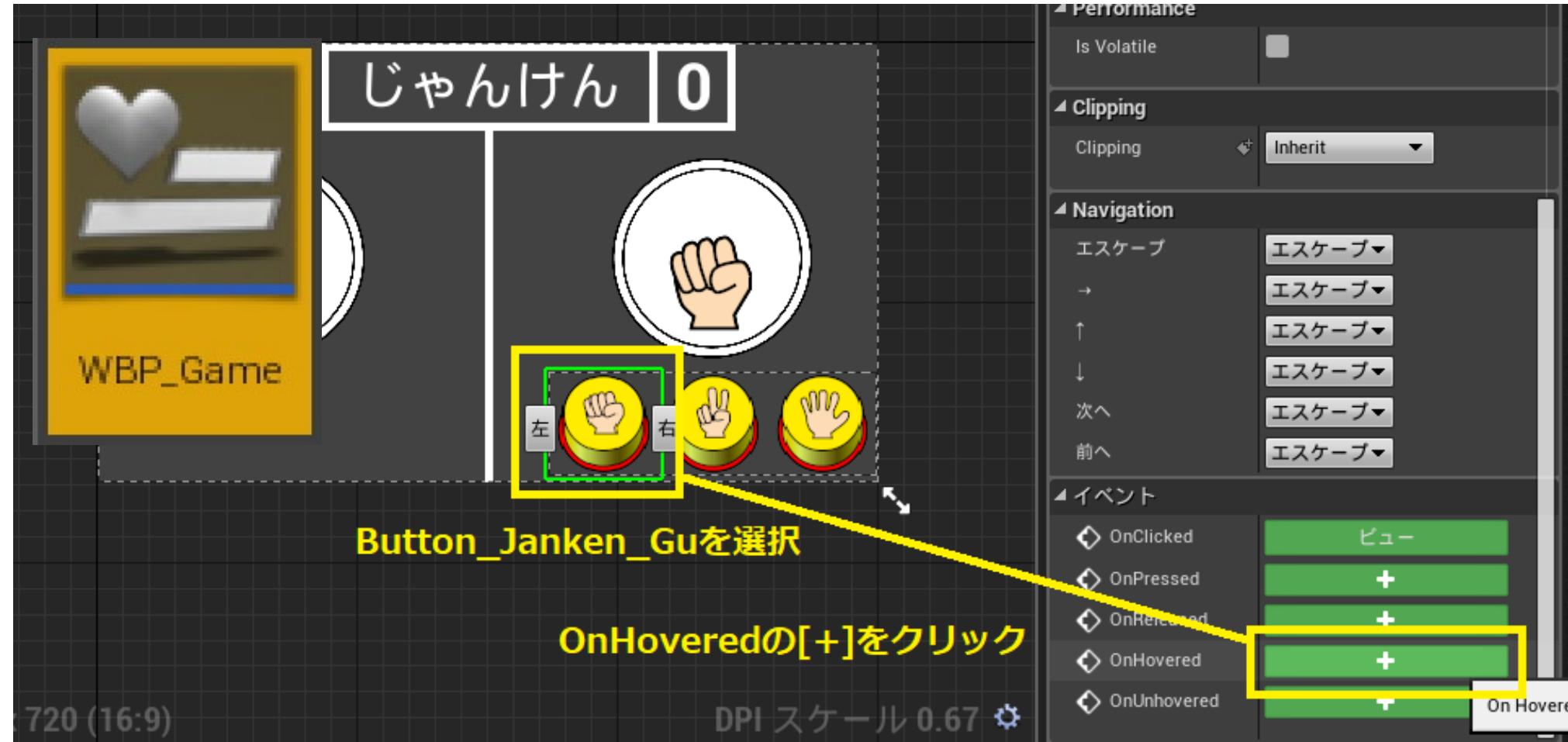
プレイして確認する



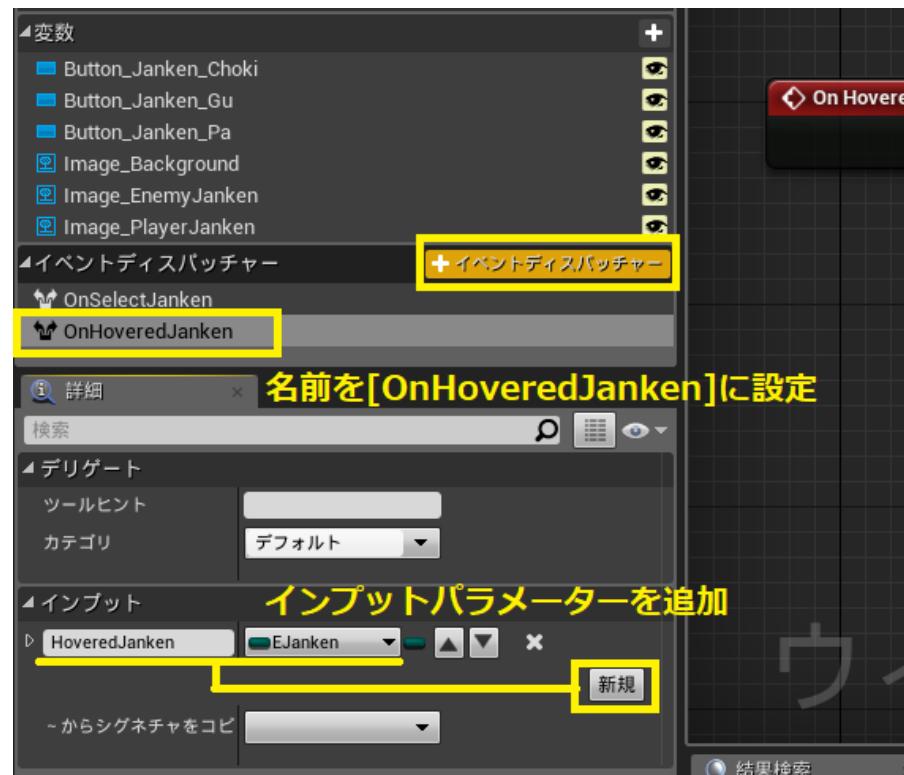
9. ウィジェットブループリントのイベントをブループリントで受け取って処理する

- 9.1 WBP_GameのButton_Janken_Guをクリックしたら、グーを選択した処理を実行するように実装する
- 9.2 Button_Janken_Choki,Button_Janken_Paも同様にクリックしたら、処理を実行するように実装する
- 9.3 MouseOverした際にプレイヤーのじゃんけん画像が変更するように実装する

Button_Janken_GuにOnHoveredを追加する 1



Button_Janken_GuにOnHoveredを追加する 2
イベントディスパッチャー：OnHoveredJankenを追加する

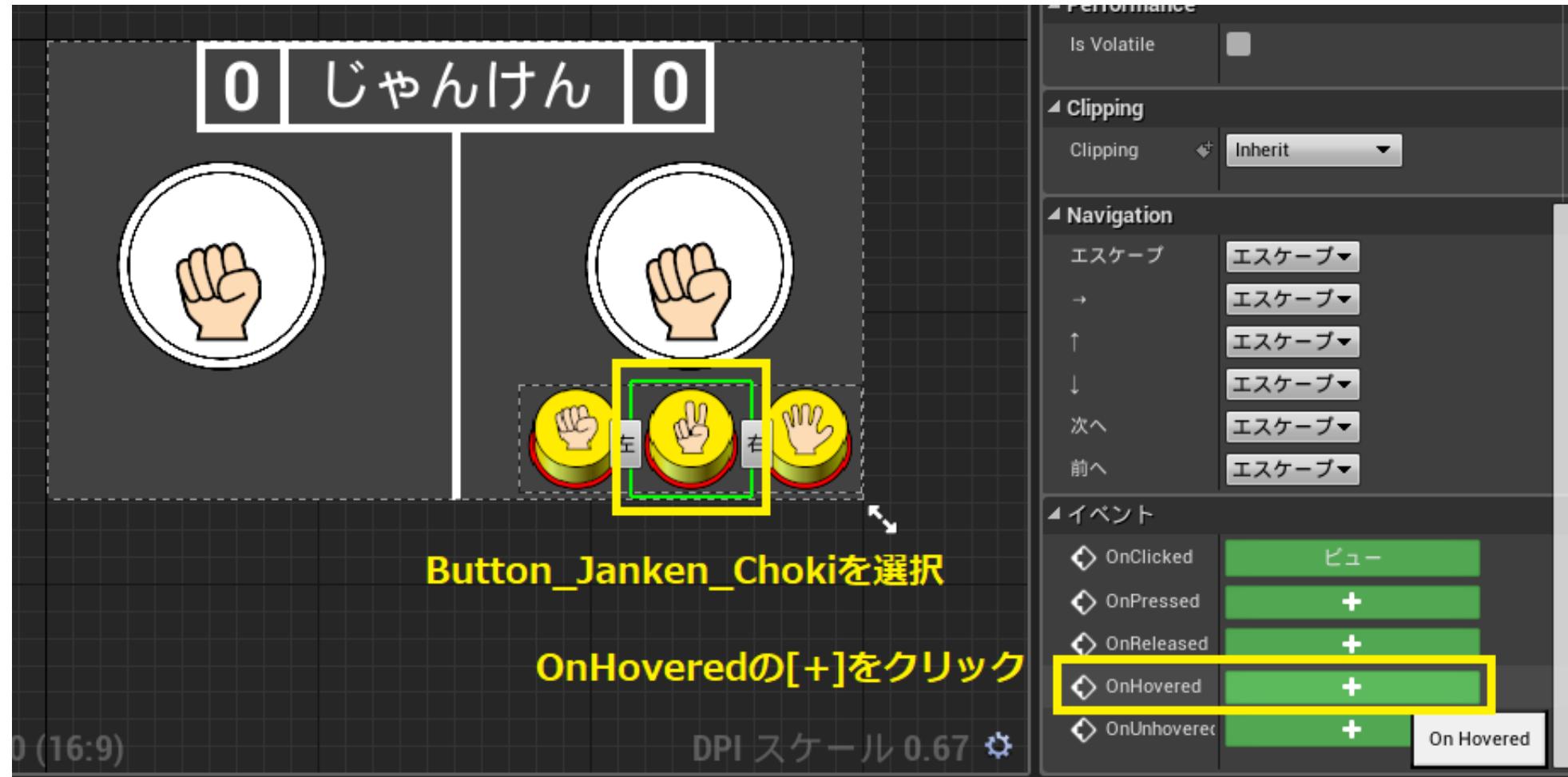


パラメータ名	型
HoveredJanken	EJanken

Button_Janken_GuにOnHoveredを追加する 3



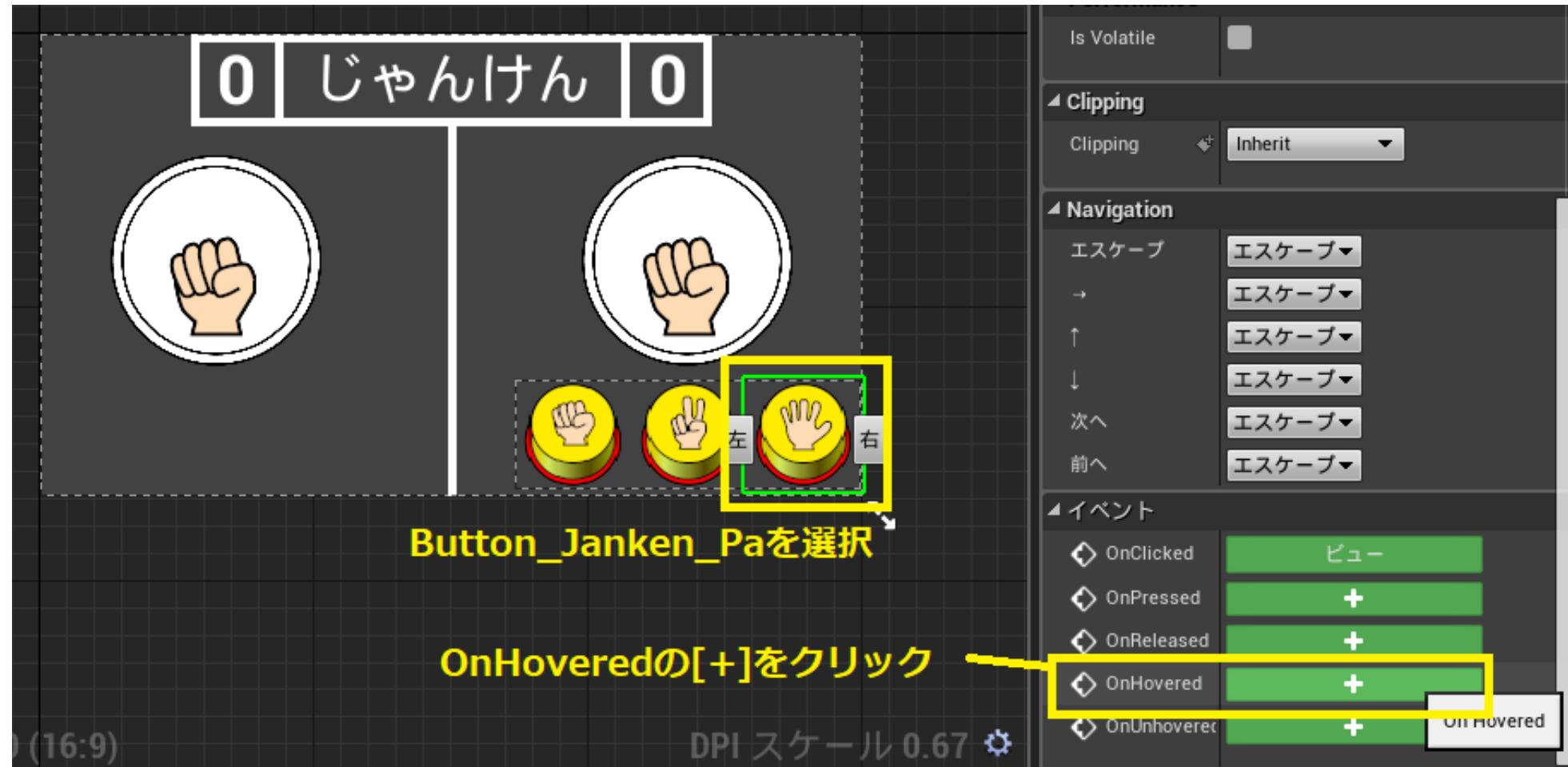
Button_Janken_ChokiにOnHoveredを追加する 1



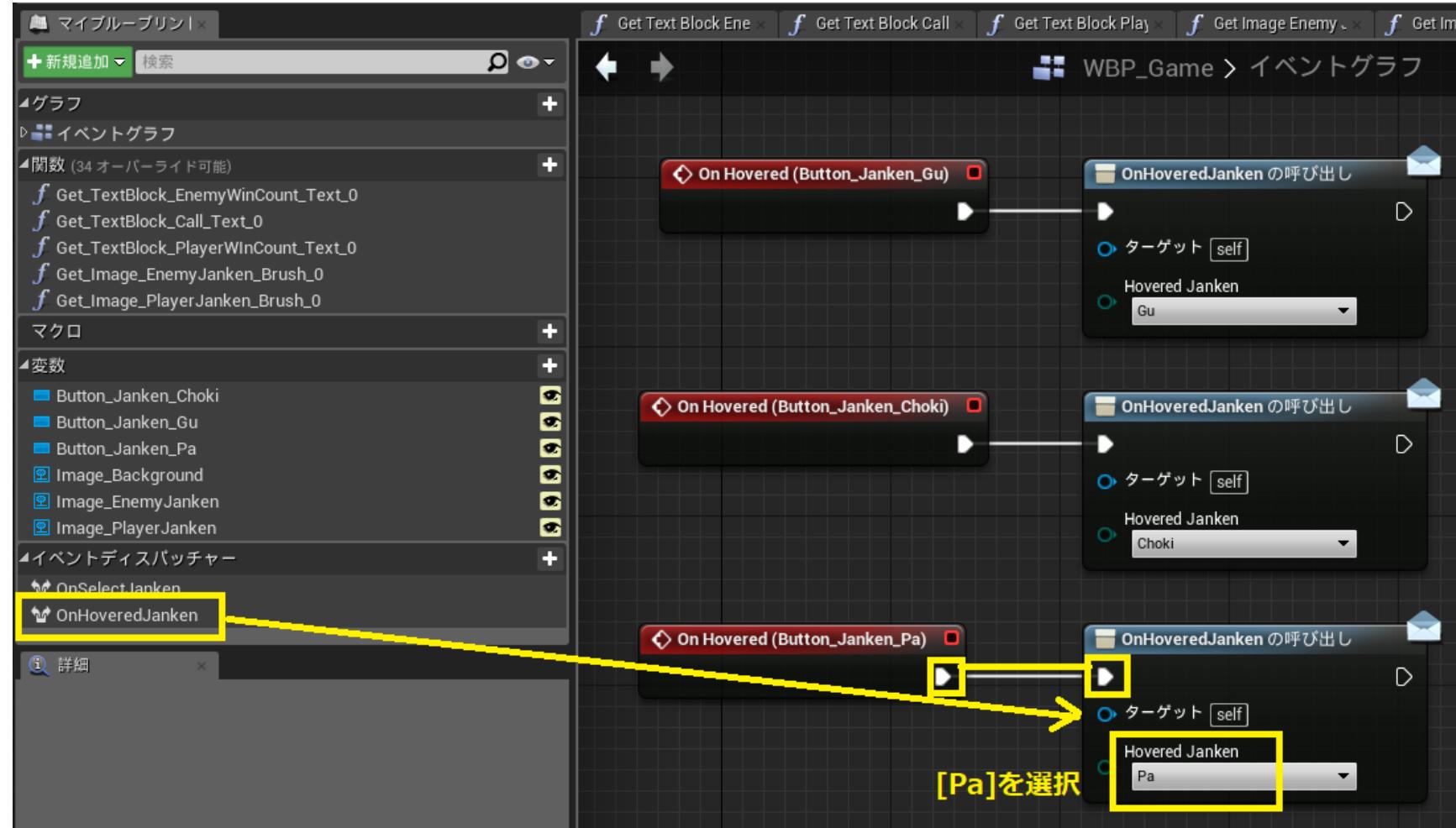
Button_Janken_ChokiにOnHoveredを追加する 2



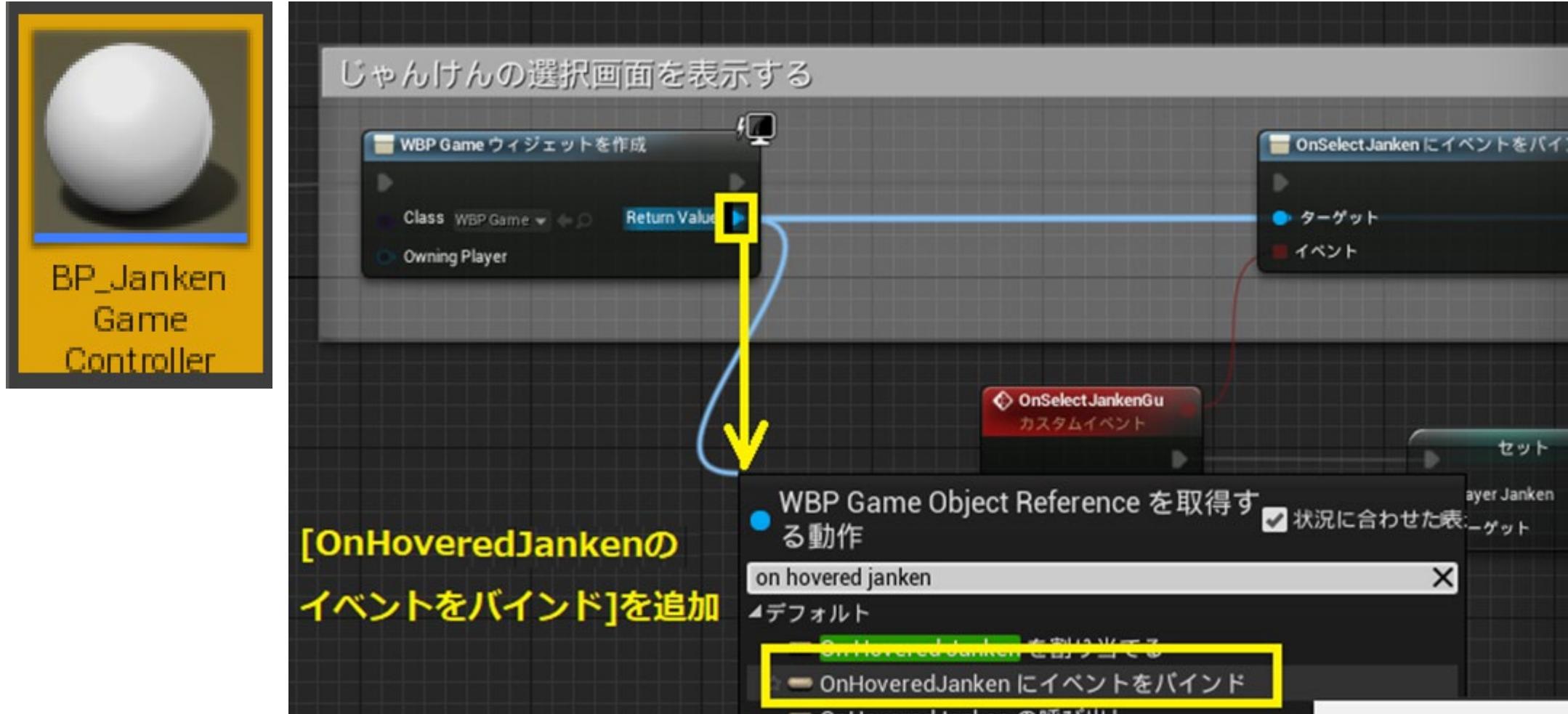
Button_Janken_PaにOnHoveredを追加する 1



Button_Janken_PaにOnHoveredを追加する 2



WBP_Game ウィジェットを作成からOnHoveredJankenのイベントをバインドを追加する 1



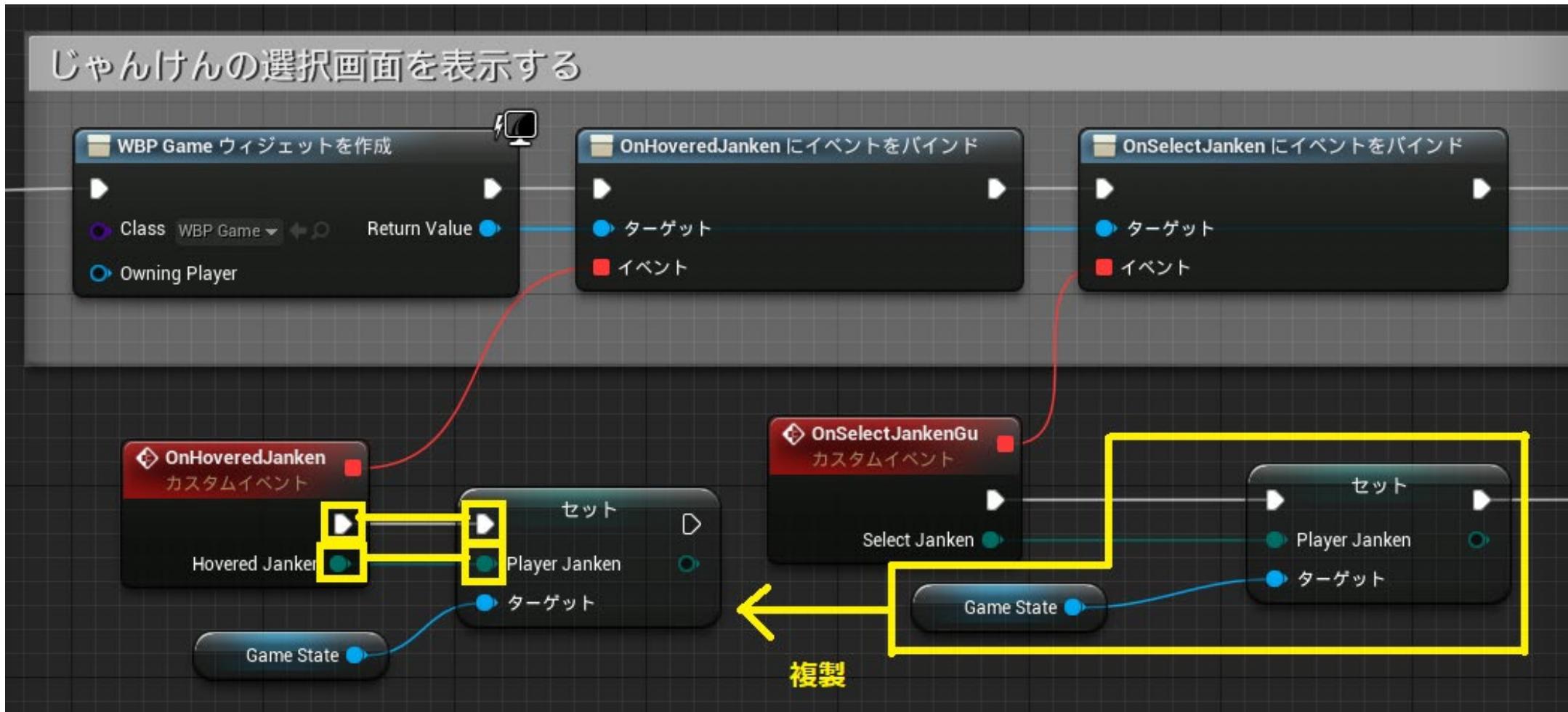
WBP_Game ウィジェットを作成から OnHoveredJanken のイベントをバインドを追加する 2



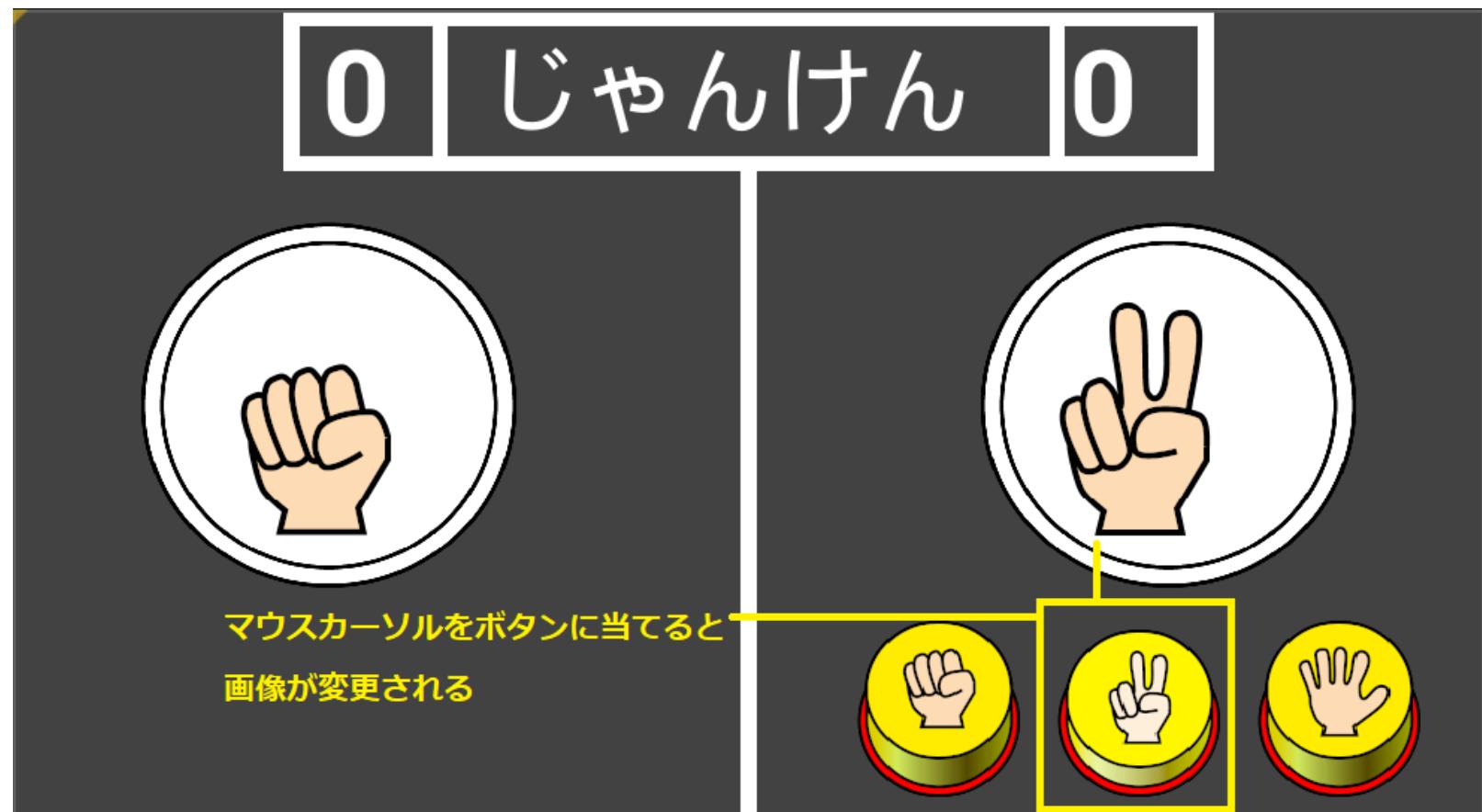
WBP_Game ウィジェットを作成から OnHoveredJanken のイベントをバインドを追加する 3



WBP_Game ウィジェットを作成から OnHoveredJanken のイベントをバインドを追加する 4



プレイして確認する



10. 結果を表示するUI作成

10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

10.6.4 WBP_JankenResultに画像を設定する関数:SetJankenResultTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：SetJankenResultsを作成

10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

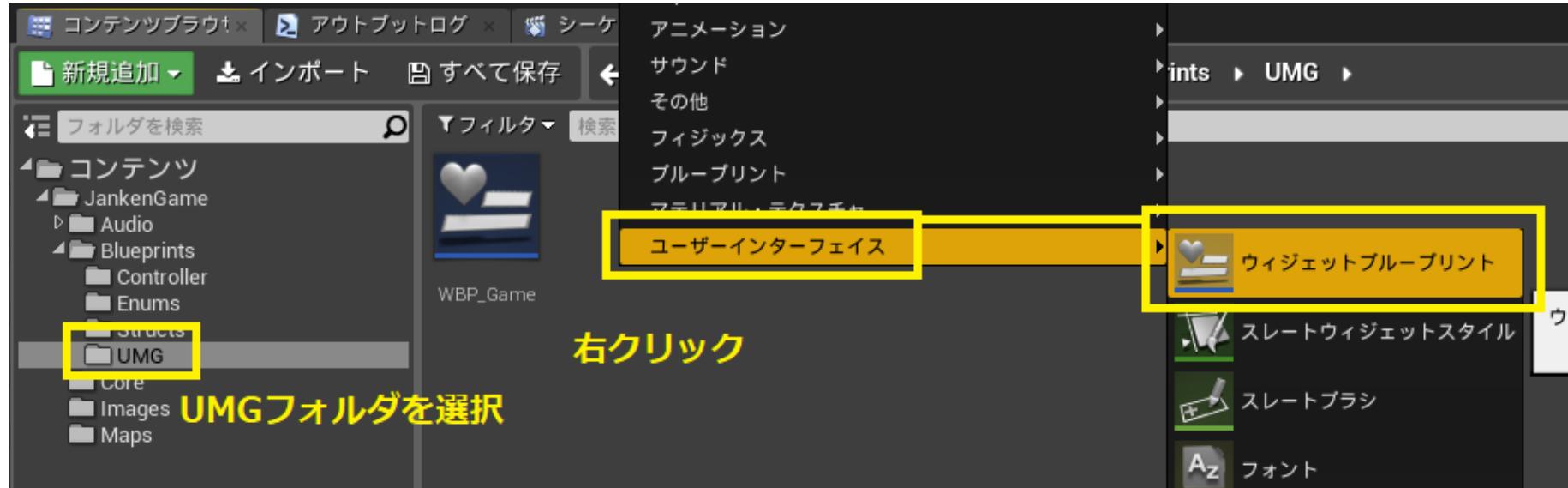
10.6.4 WBP_JankenResultに画像を設定する関数:SetJankenResultTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：SetJankenResultsを作成

10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

ウィジェットブループリント WBP_GameResultを作成する



10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

10.6.4 WBP_JankenResultに画像を設定する関数:SetJankenResultTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：SetJankenResultsを作成

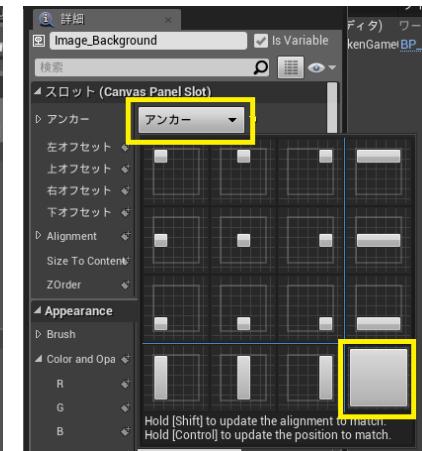
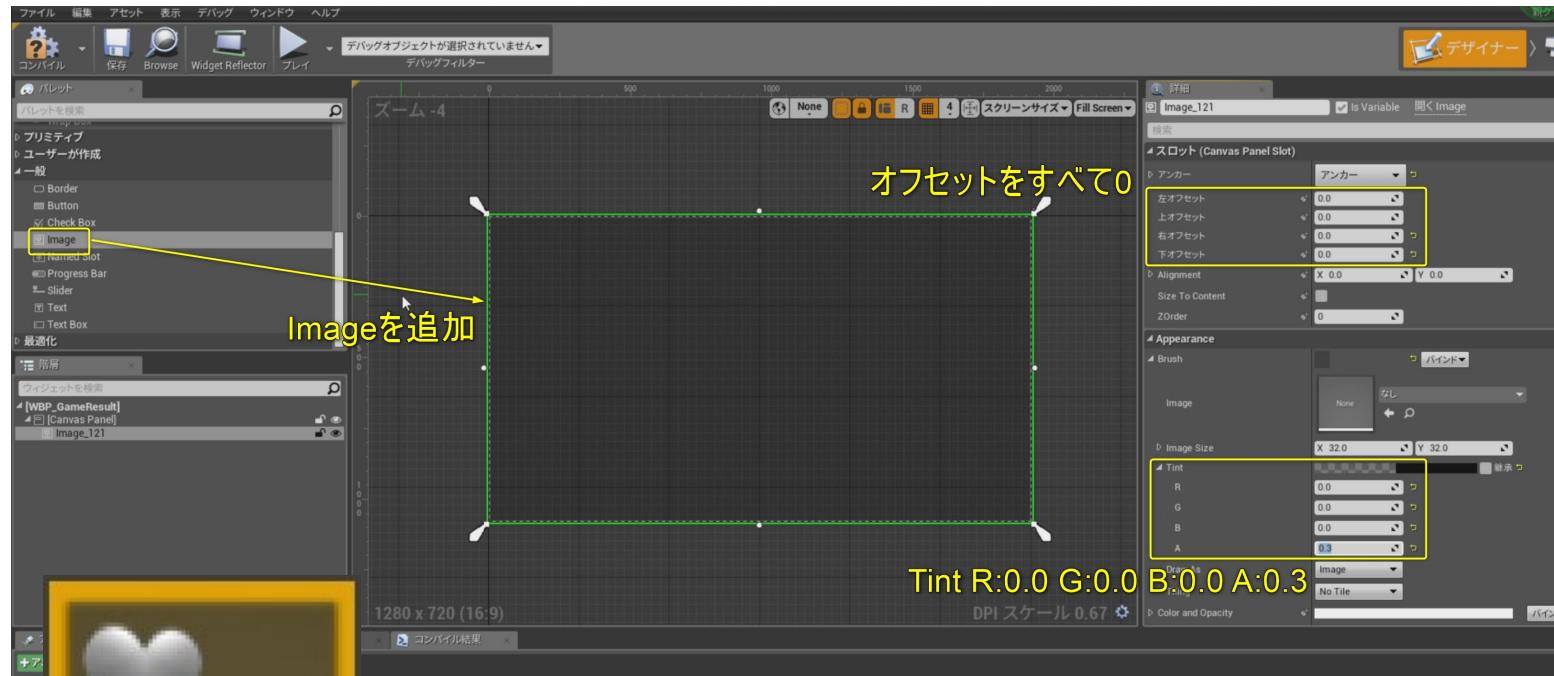
10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

UI作成完成イメージ

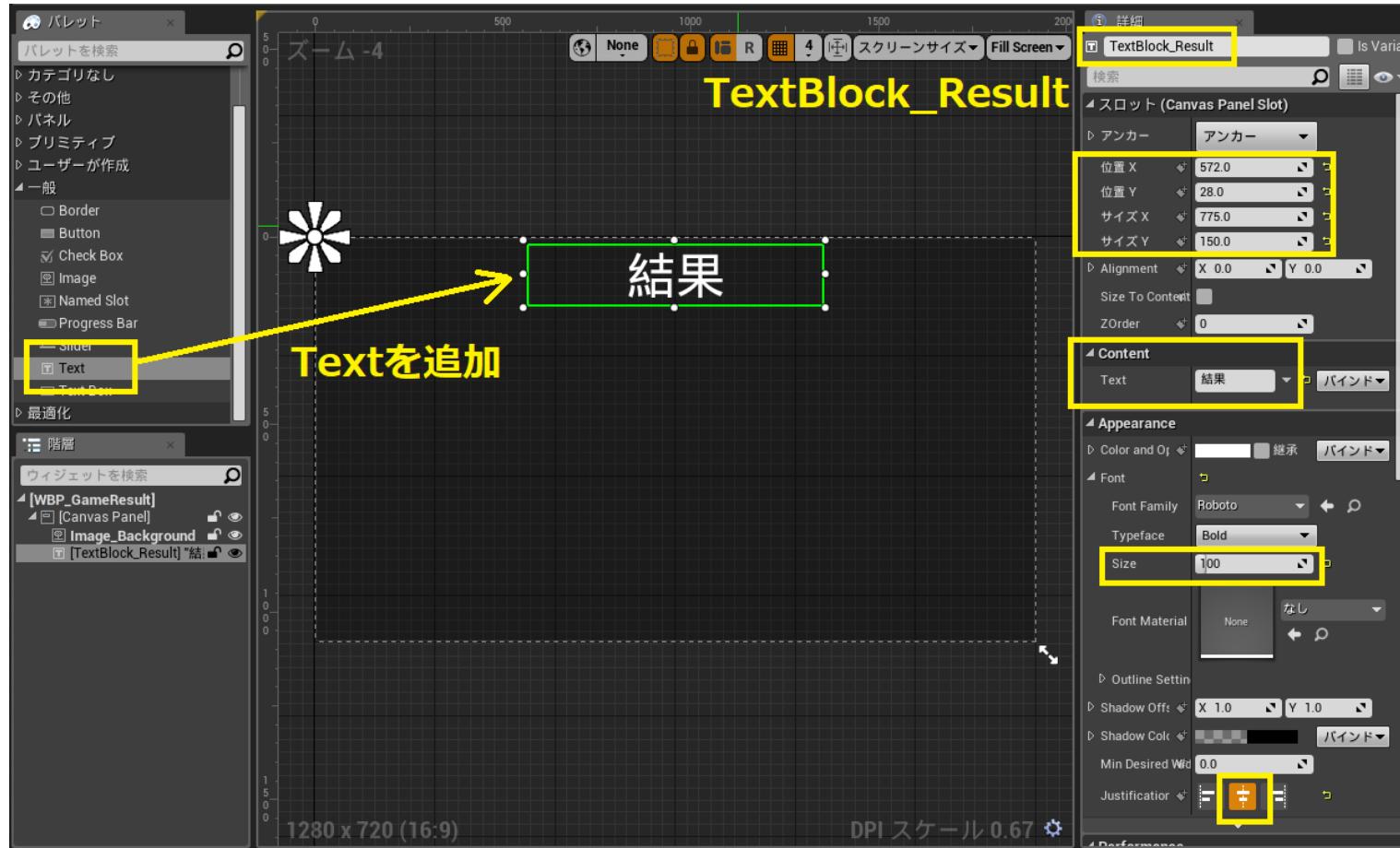


Image:背景の設定



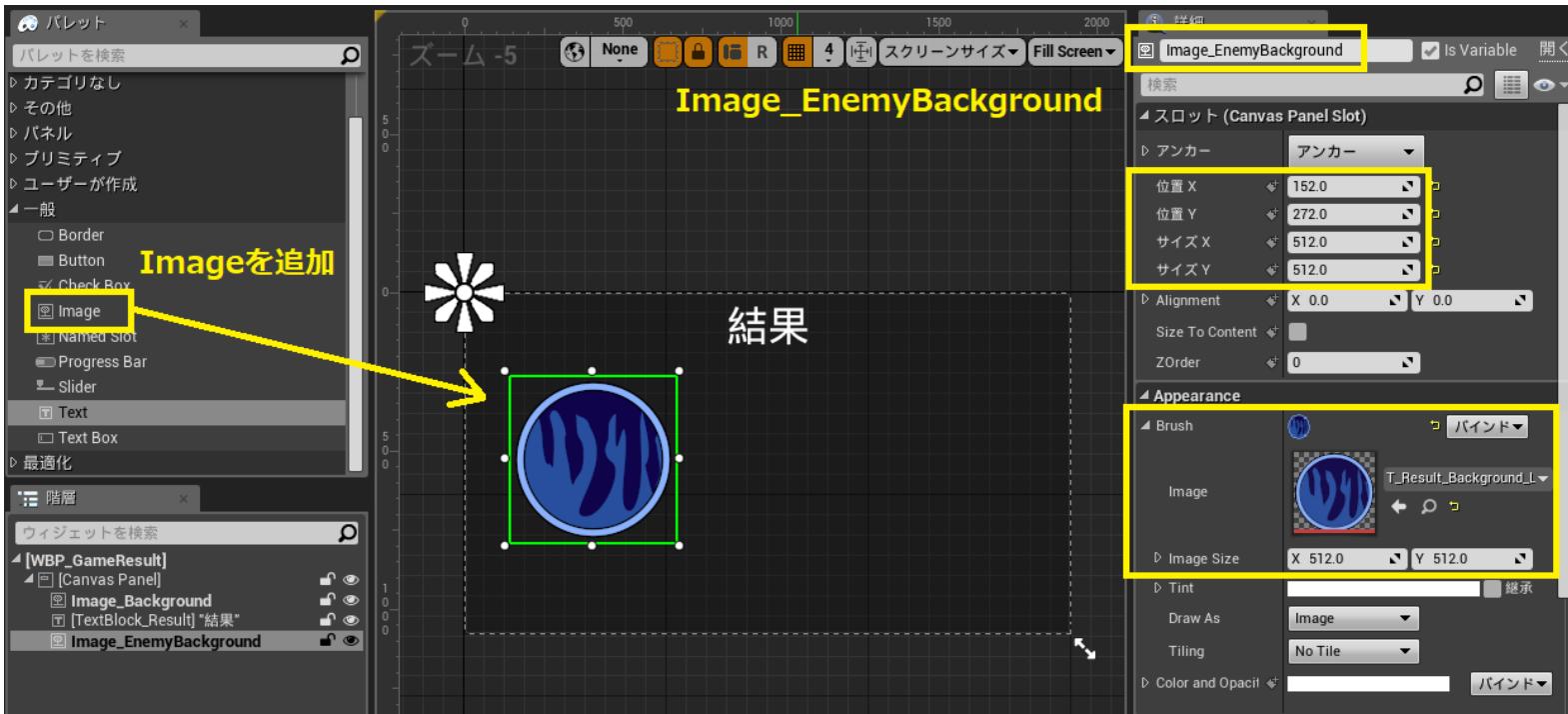
プロパティ	値
左オフセット	0.0
上オフセット	0.0
右オフセット	0.0
下オフセット	0.0
Brush > Tint	R:0.0 G:0.0 B:0.0 A:0.3

Text: TextBlock_Result の設定



プロパティ	値
変数名	TextBlock_Result
位置X	572.0
位置Y	28.0
サイズX	775.0
サイズY	150.0
Text	結果
Font > Size	100
Justification	テキスト中央揃え

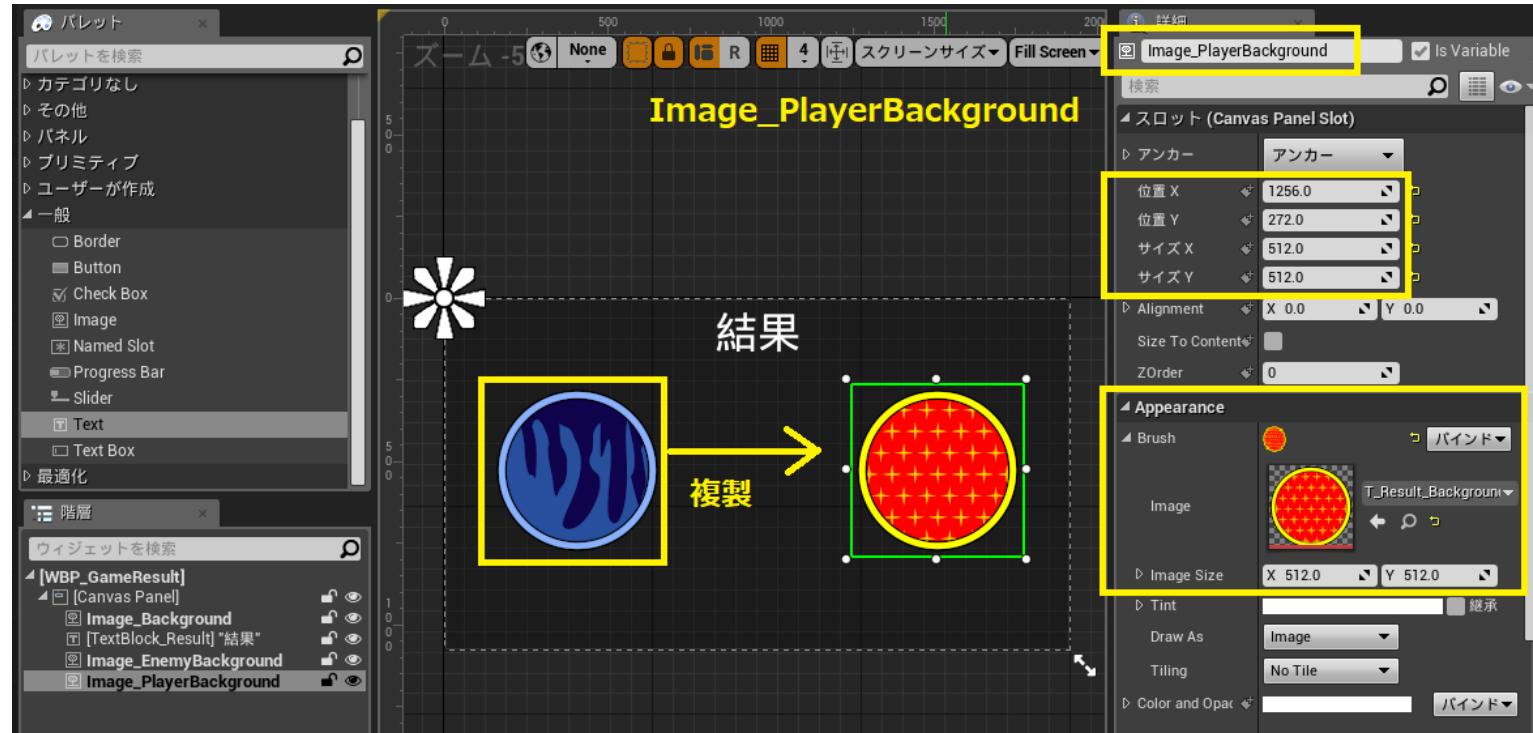
Image:Image_EnemyBackgroundを追加する



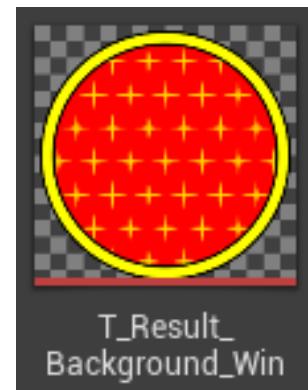
プロパティ	値
変数名	Image_EnemyBackground
位置X	152.0
位置Y	272.0
サイズX	512.0
サイズY	512.0
Brush > Image	T_Result_Background_Lose



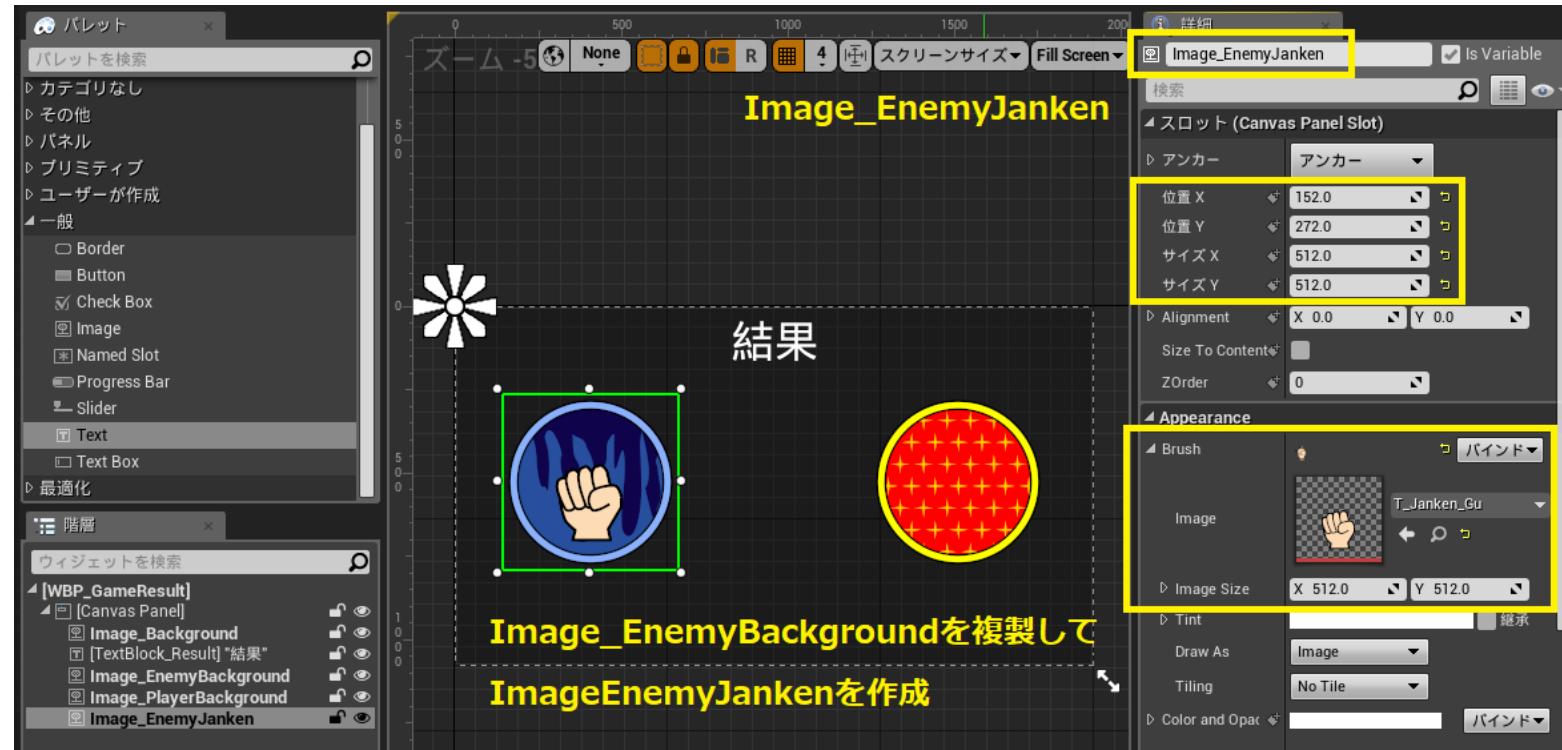
Image_EnemyBackgroundを複製して、 Image_PlayerBackgroundを作成する



プロパティ	値
変数名	Image_PlayerBackground
位置X	1256.0
位置Y	272.0
サイズX	512.0
サイズY	512.0
Brush > Image	T_Result_Background_Win



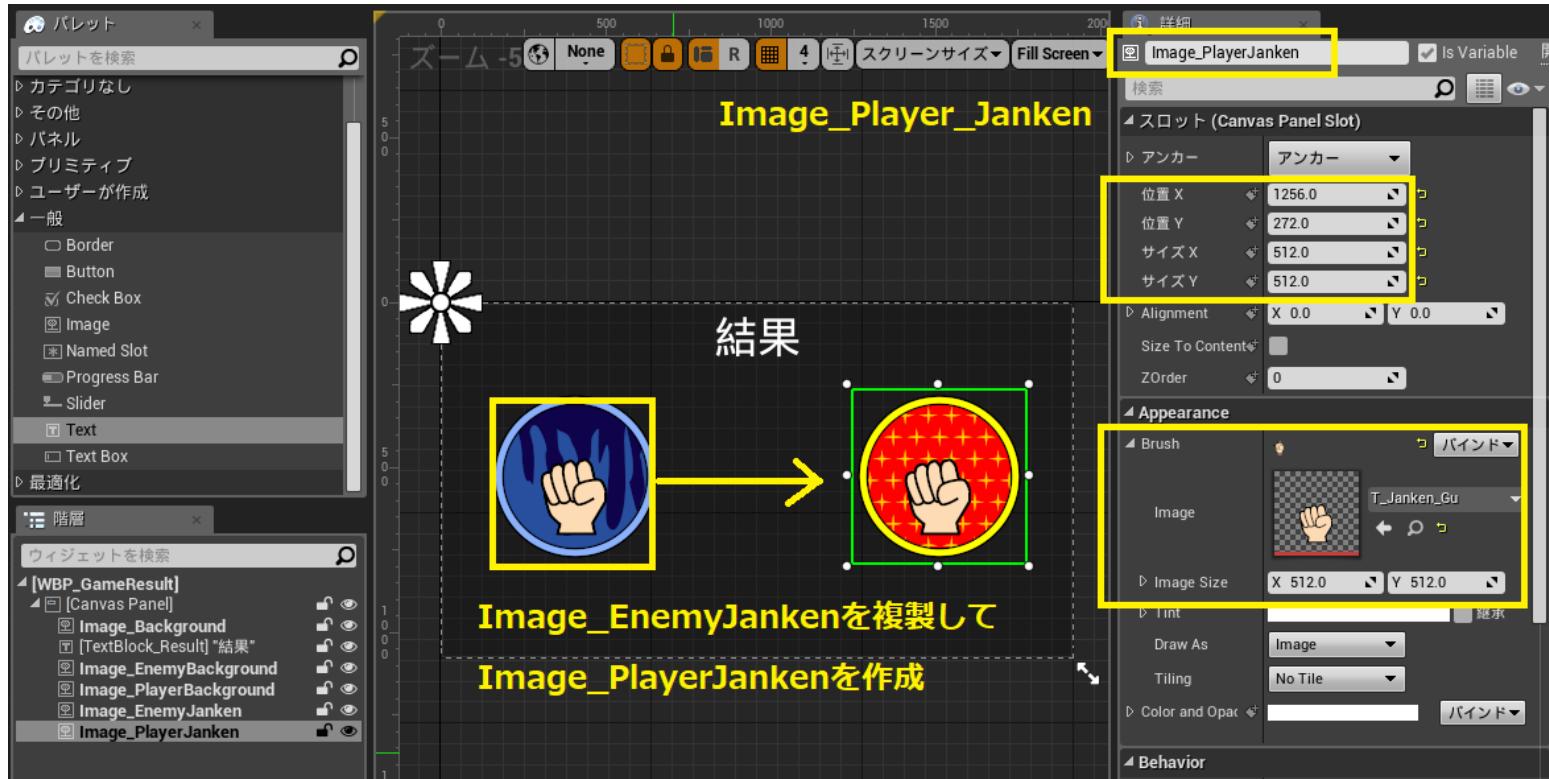
Image_EnemyBackgroundを複製して、 Image_EnemyJankenを作成する



プロパティ	値
変数名	Image_EnemyJanken
位置X	152.0
位置Y	272.0
サイズX	512.0
サイズY	512.0
Brush > Image	T_Janken_Gu



Image_EnemyJankenを複製して、 Image_PlayerJankenを作成する



プロパティ	値
変数名	Image_PlayerJanken
位置X	1256.0
位置Y	272.0
サイズX	512.0
サイズY	512.0
Brush > Image	T_Janken_Gu



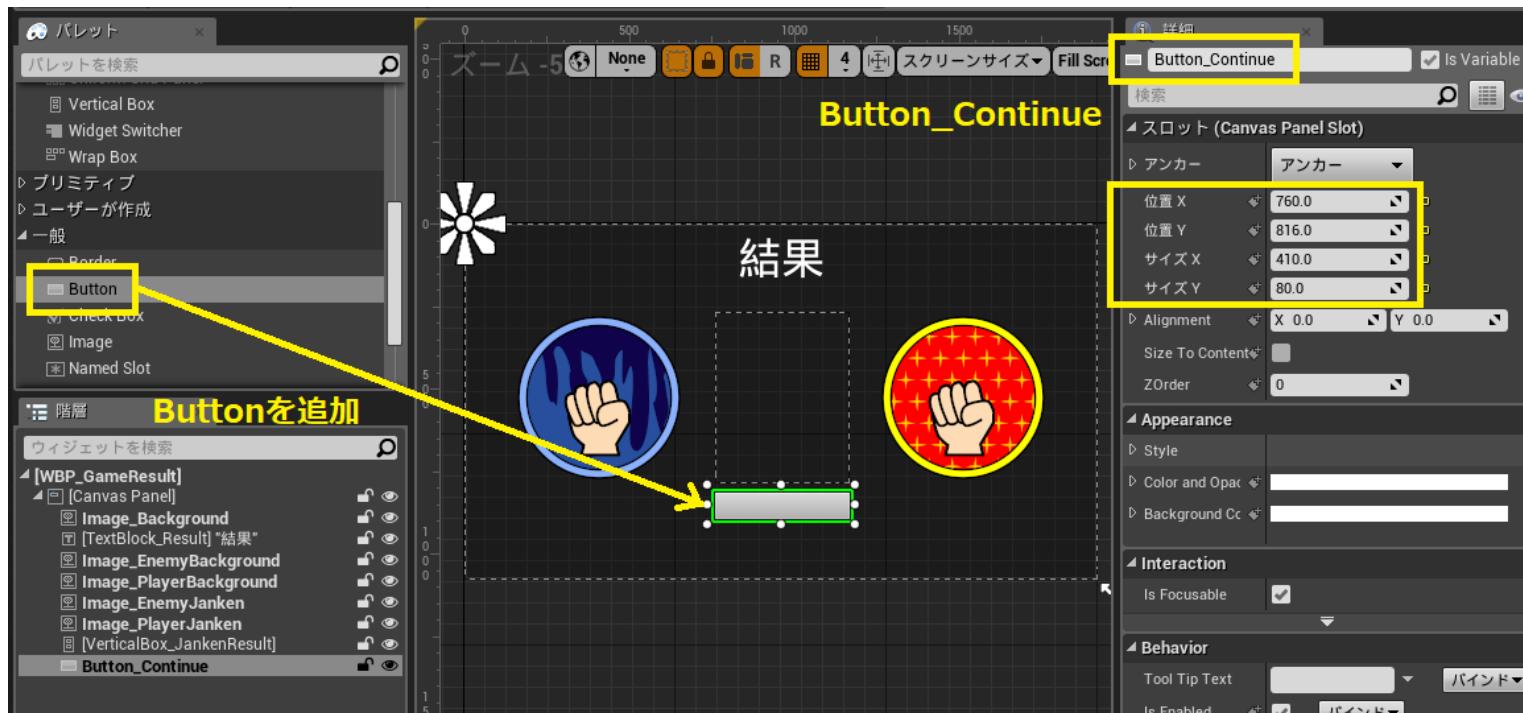
VerticalBox:VerticalBox_JankenResultを追加する

The screenshot shows the Unreal Engine 4 Editor interface. On the left, the 'Palette' panel is open, with the 'Vertical Box' item selected and highlighted by a yellow box. A large yellow arrow points from the palette towards the center canvas area. The center canvas contains two circular images representing hands: one blue hand on the left and one red hand on the right, both set against a dark background. Above these images, the word '結果' (Result) is displayed in white text. To the right of the canvas is the 'Details' panel, which is titled 'VerticalBox_JankenResult'. The 'Slot' section of the details panel is expanded, showing the following properties:

プロパティ	値
変数名	VerticalBox_JankenResult
位置X	760.0
位置Y	268.0
サイズX	410.0
サイズY	500.0

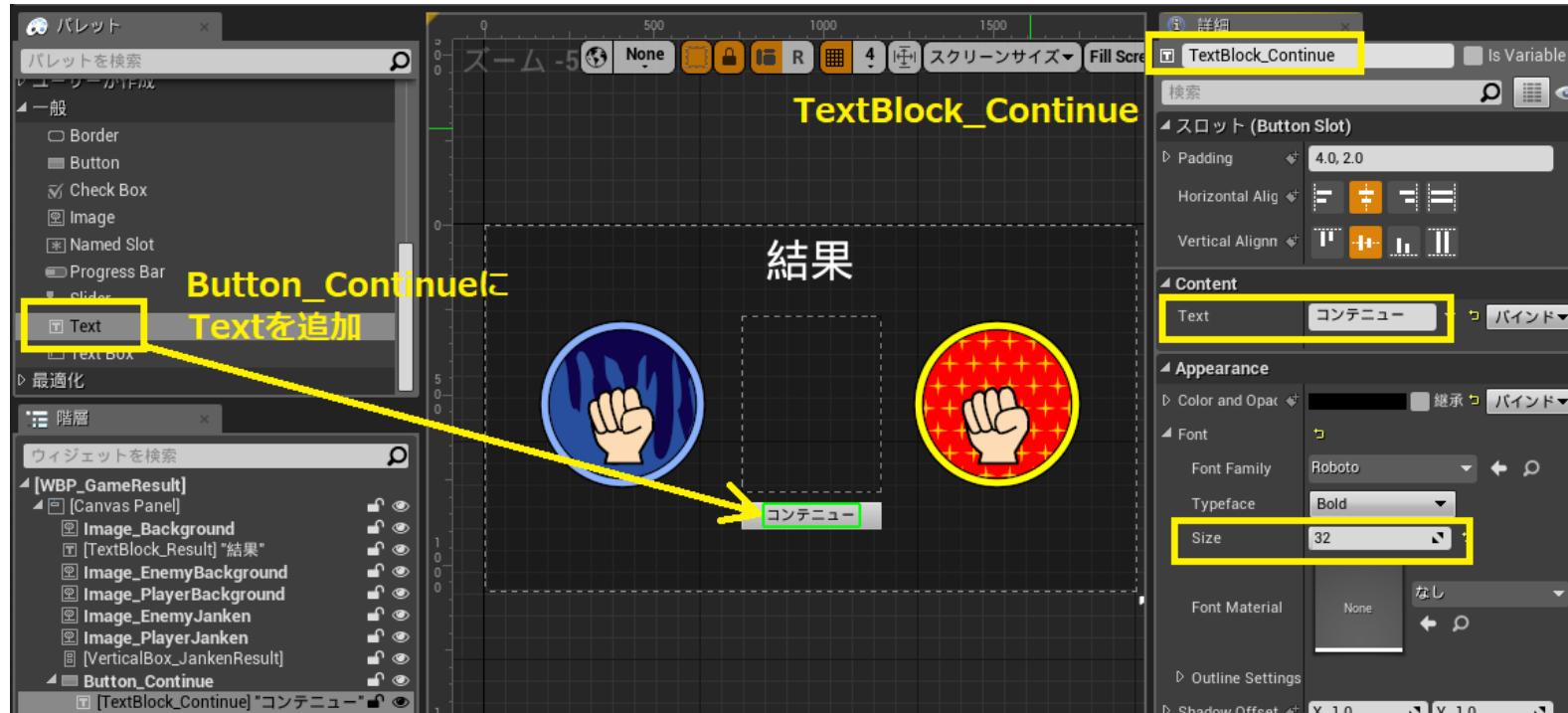
The 'Behavior' and 'Render Transform' sections are also visible in the details panel.

Button:Button_Continueを追加する



プロパティ	値
変数名	Button_Continue
位置X	760.0
位置Y	816.0
サイズX	410.0
サイズY	80

Button_ContinueにText:TextBlock_Continueを追加する



プロパティ	値
変数名	TextBlock_Continue
Text	コンティニュー
Color and Opacity	黒(R:0.0, G:0.0 B:0.0 A:1.0)
Font > Size	32

Button_Continueを複製して、 Button_LoadTitleを作成する



プロパティ	値
変数名	Button_LoadTitle
位置X	760.0
位置Y	928.0
サイズX	410.0
サイズY	80

Button_LoadTitleのText:TextBlock_LoadTitleを変更する



プロパティ	値
変数名	TextBlock_LoadTitle
Text	タイトル

10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

10.6.4 WBP_JankenResultに画像を設定する関数:SetJankenResultTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：SetJankenResultsを作成

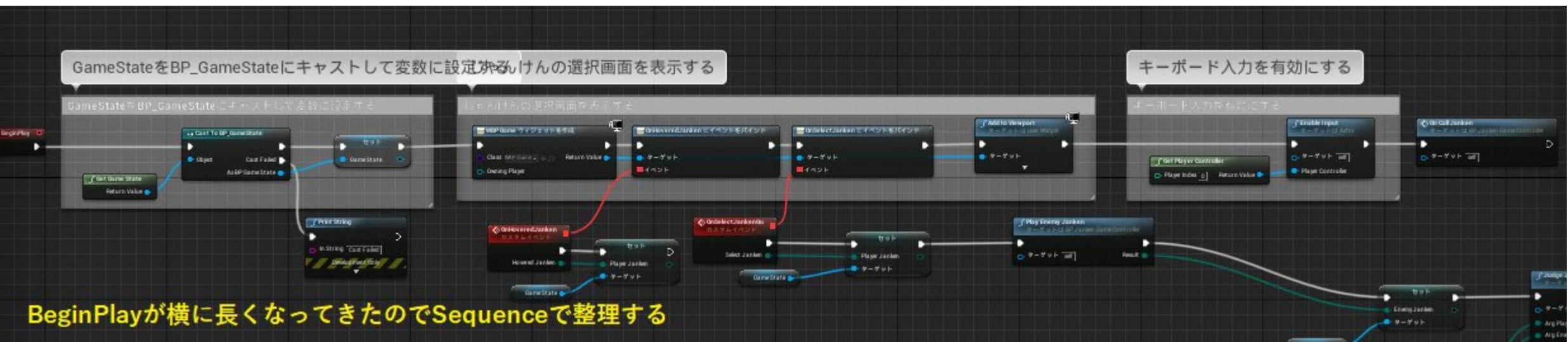
10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

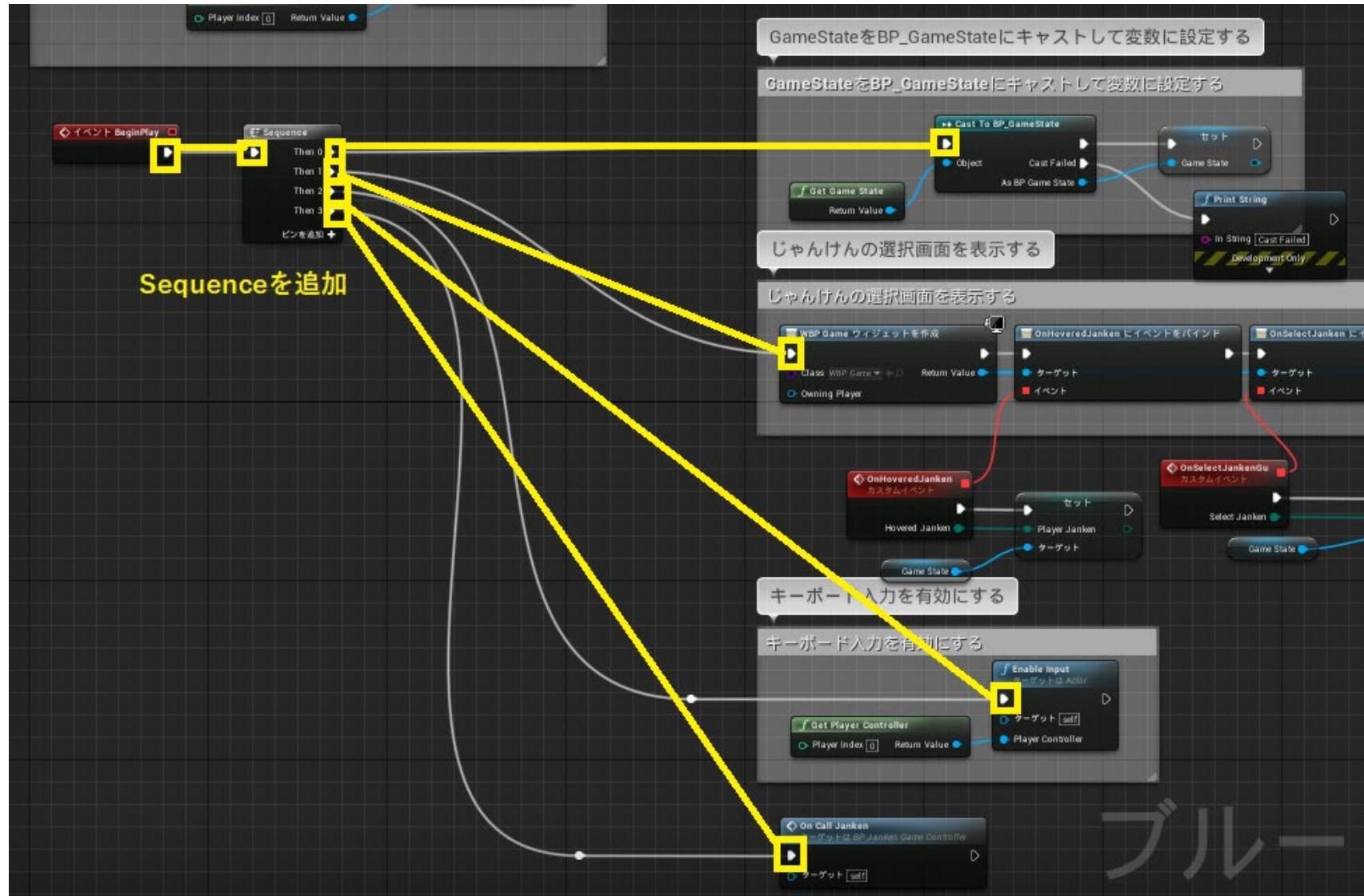
BeginPlayをSequenceで整理する 1



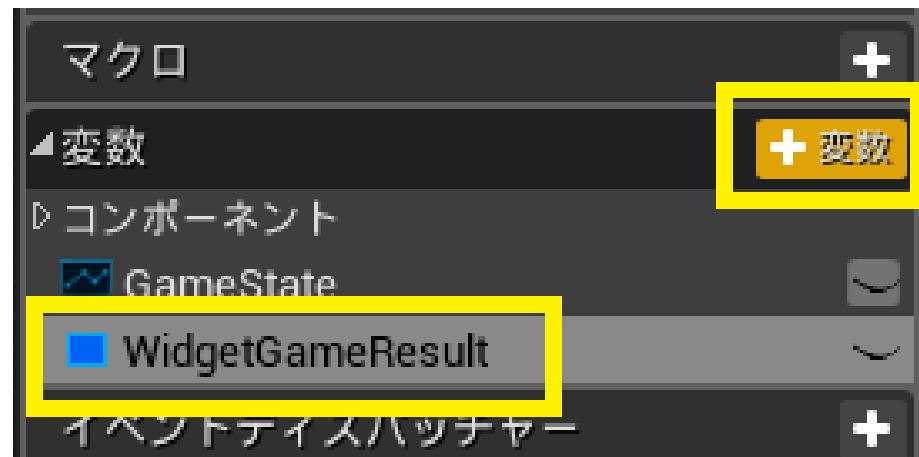
GameStateをBP_GameStateにキャストして変数に設定する
いわゆるひのき選択画面を表示する



BeginPlayをSequenceで整理する 2

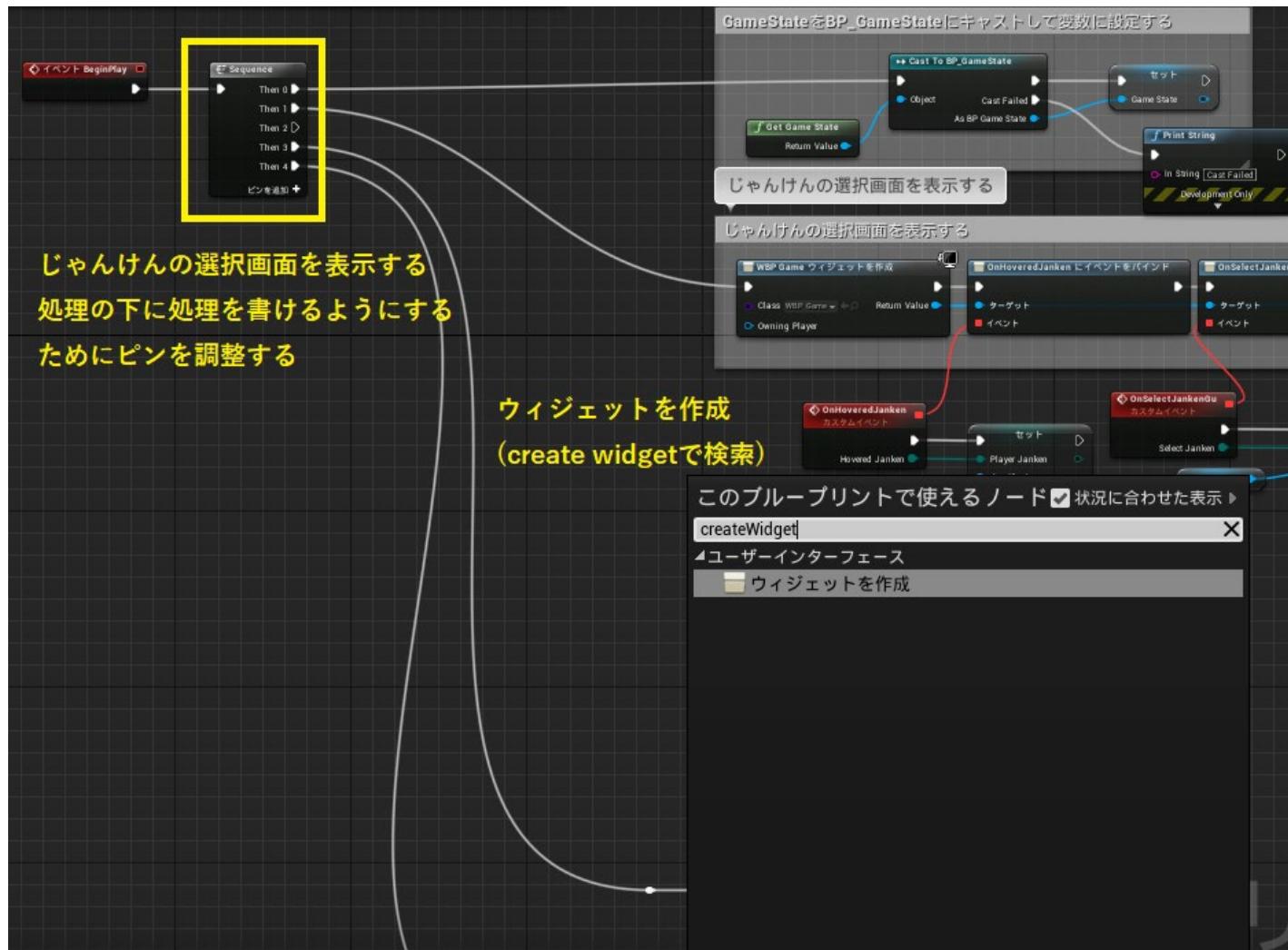


変数を追加する

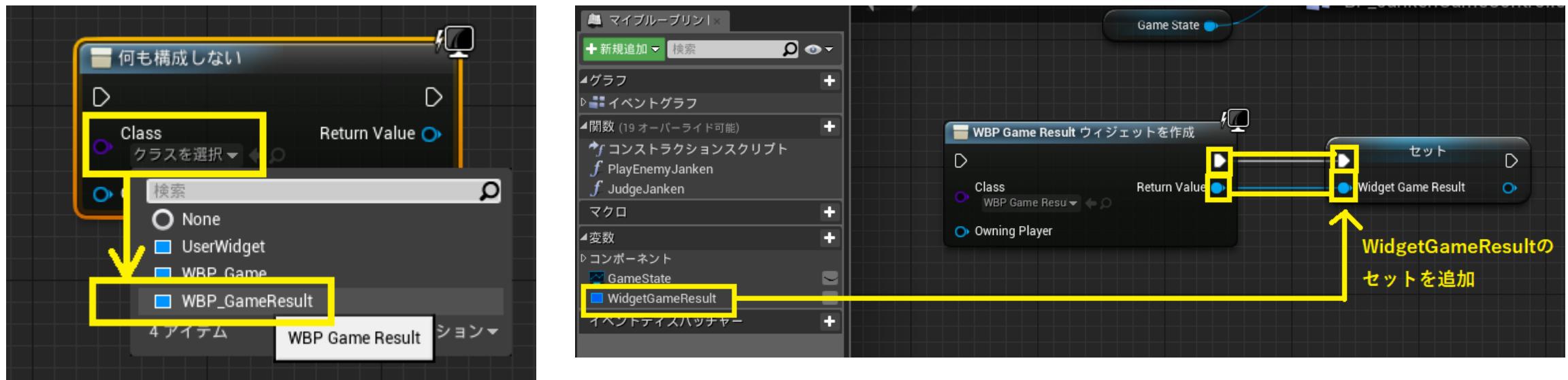


変数名	変数の型	デフォルト値
GameState	BP_GameState	-(設定なし)
WidgetGameResult	WBP_GameResult (Object Reference)	-(設定なし)

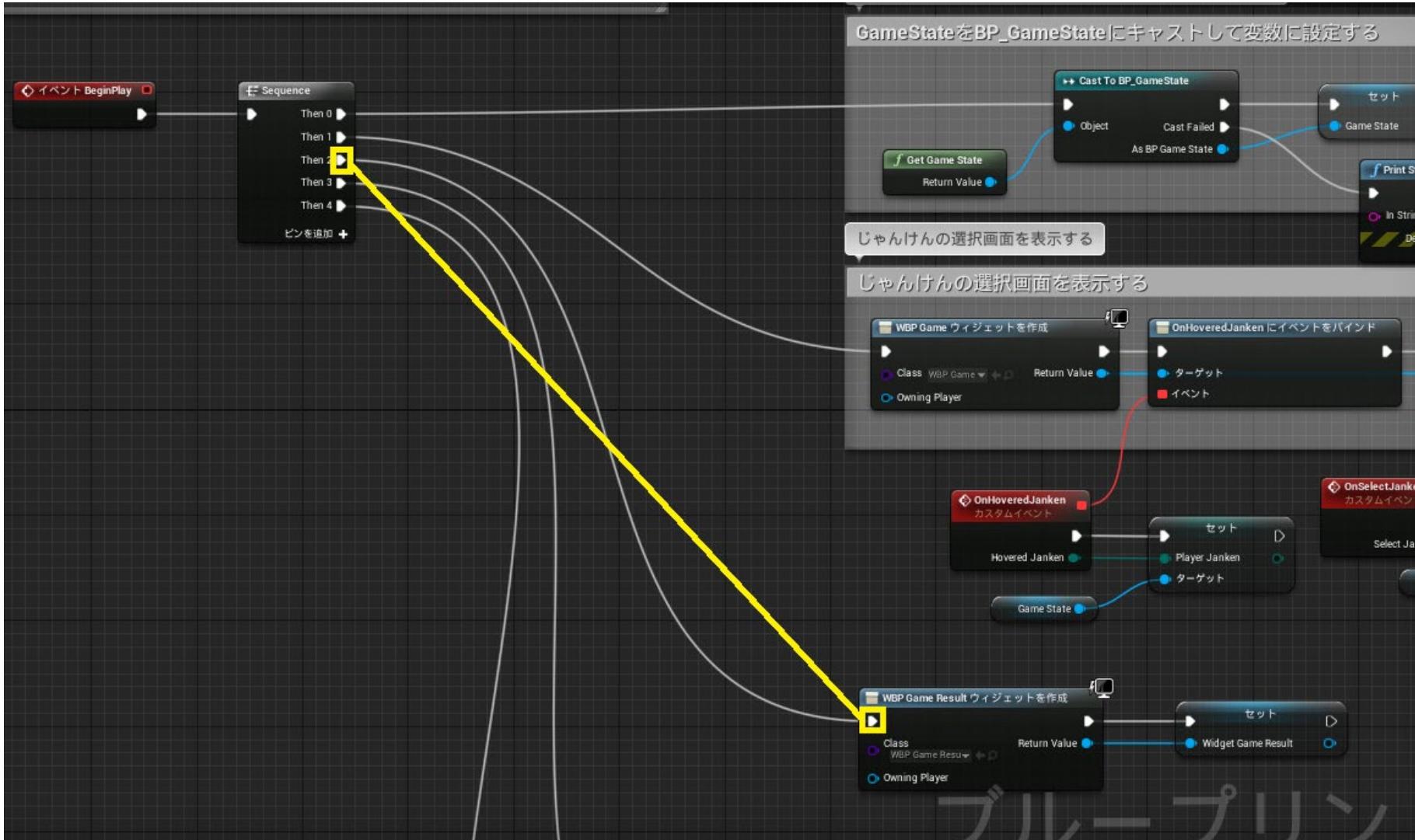
WBP_Resultのウィジェットを表示する 1



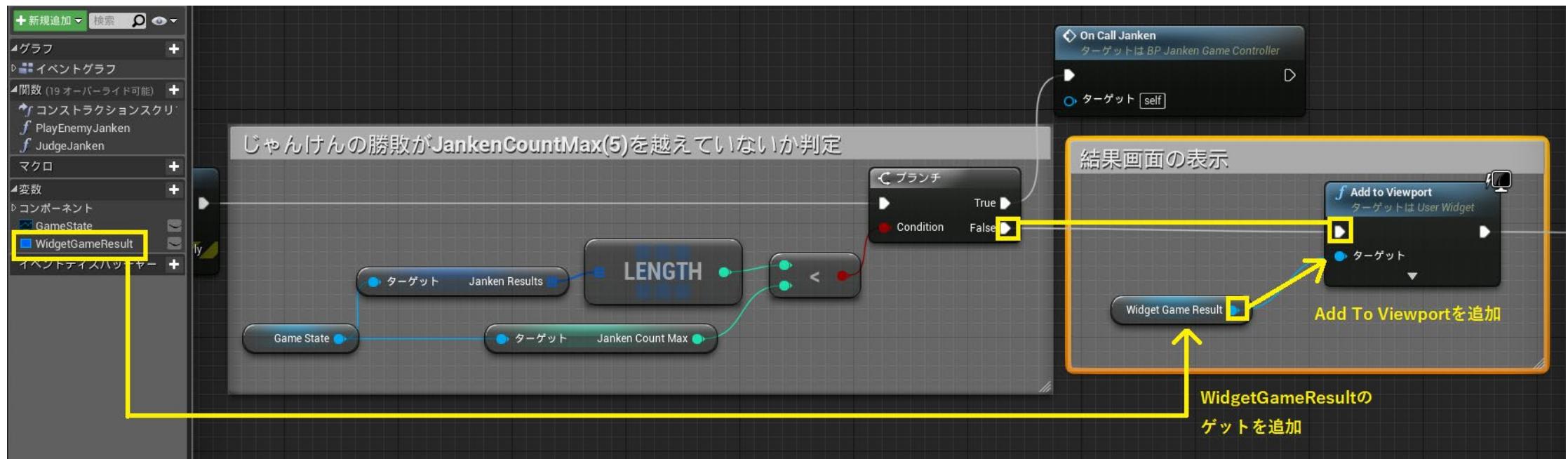
WBP_Resultのウィジェットを表示する 2



WBP_Resultのウィジェットを表示する 4



WBP_Resultのウィジェットを表示する 5



プレイして確認する



10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

10.6.4 WBP_JankenResultに画像を設定する関数:SetJankenResultTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：SetJankenResultsを作成

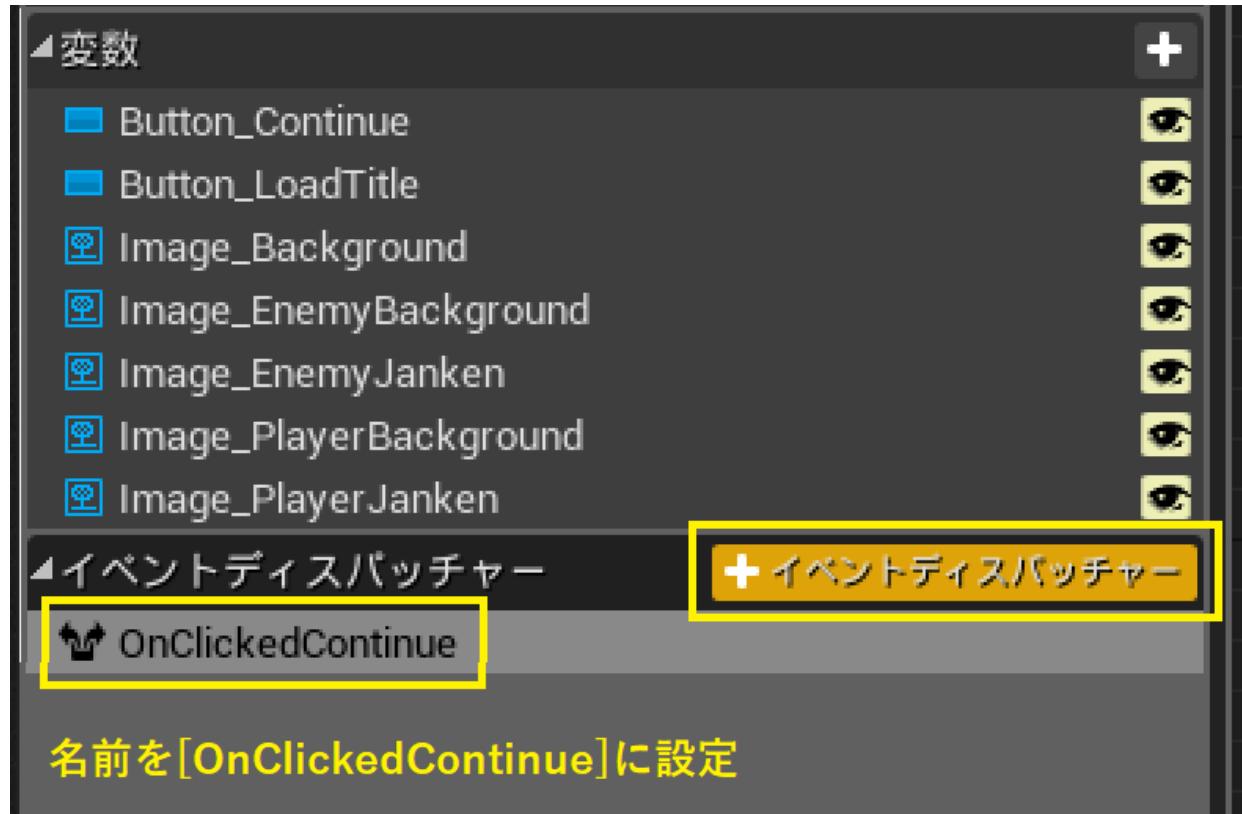
10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

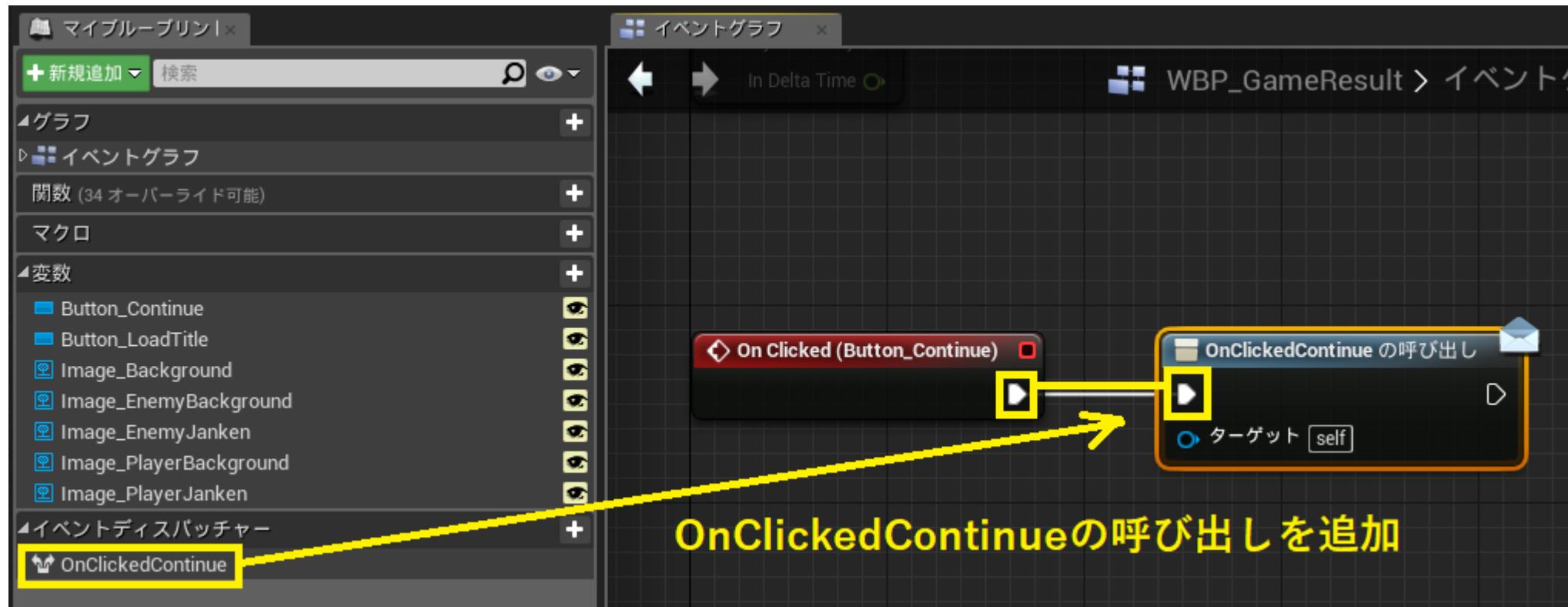
Button_ContinueにOnClickedイベントを追加する 1



WBP_Resultのウィジェットを表示する 2 イベントディスパッチャー:OnClickedContinueを追加する



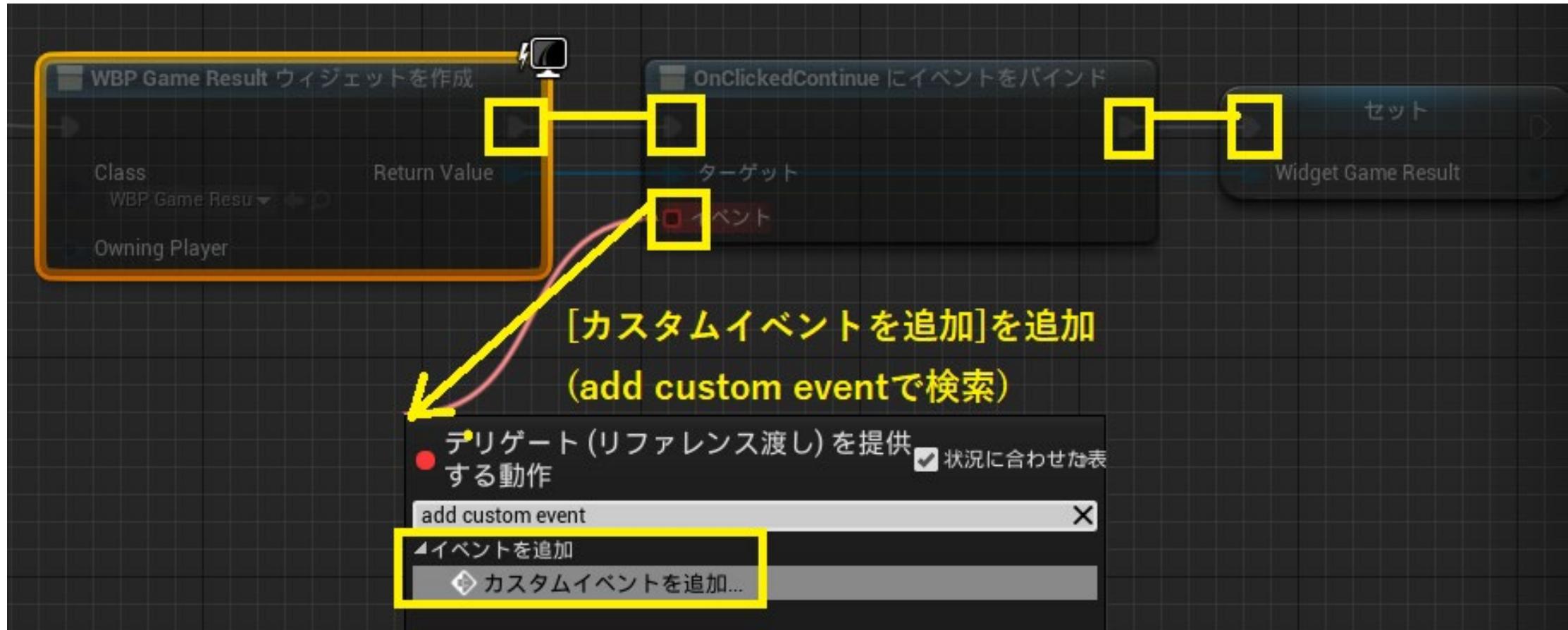
WBP_Resultのウィジェットを表示する 3



WBP_GameResultのOnClickedContinueにイベントをバインドする 1



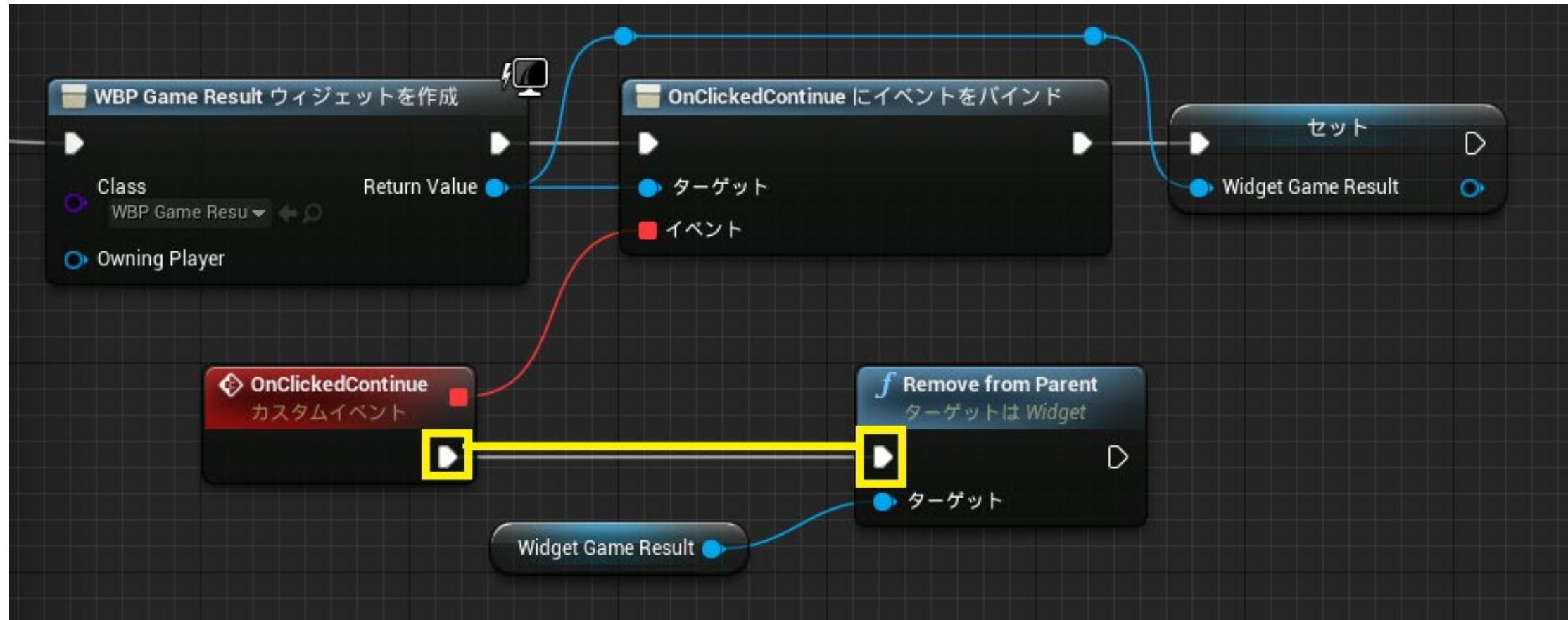
WBP_GameResultのOnClickedContinueにイベントをバインドする 2



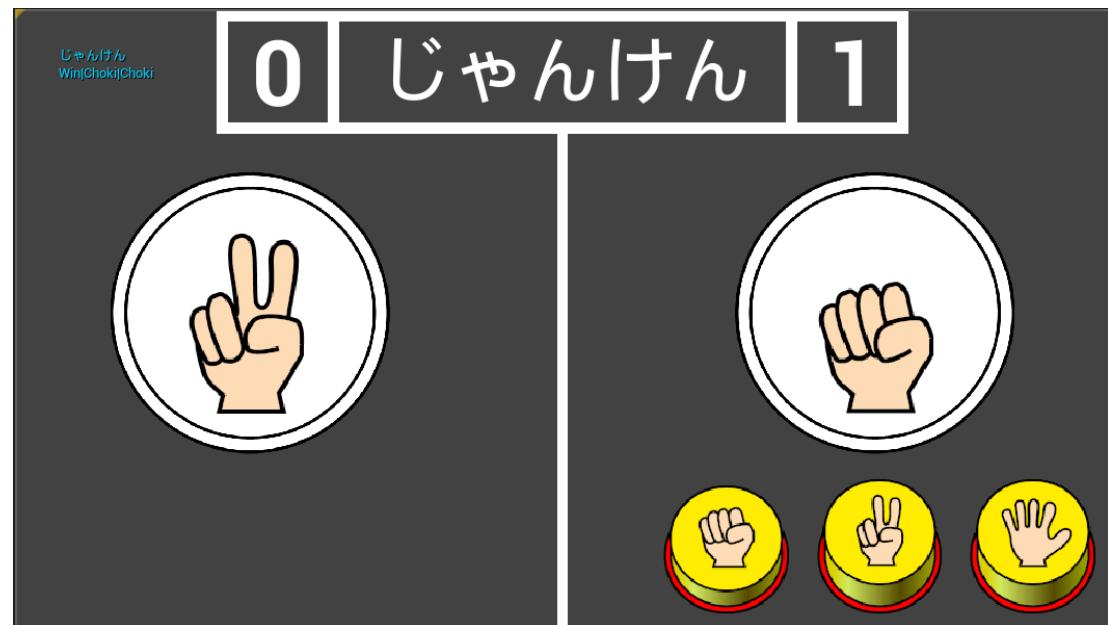
WBP_GameResultのOnClickedContinueにイベントをバインドする 3



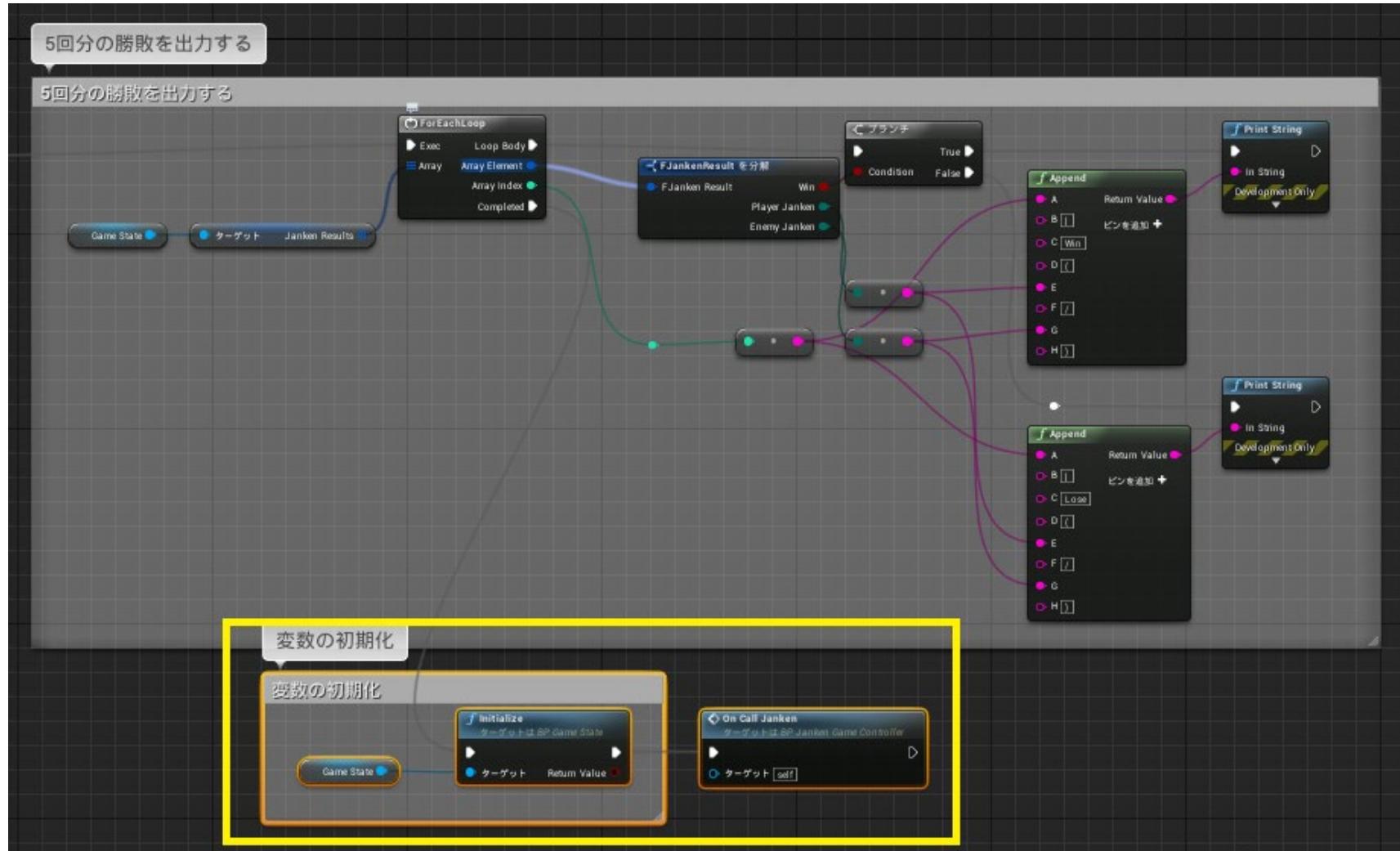
WBP_GameResultのOnClickedContinueにイベントをバインドする 4



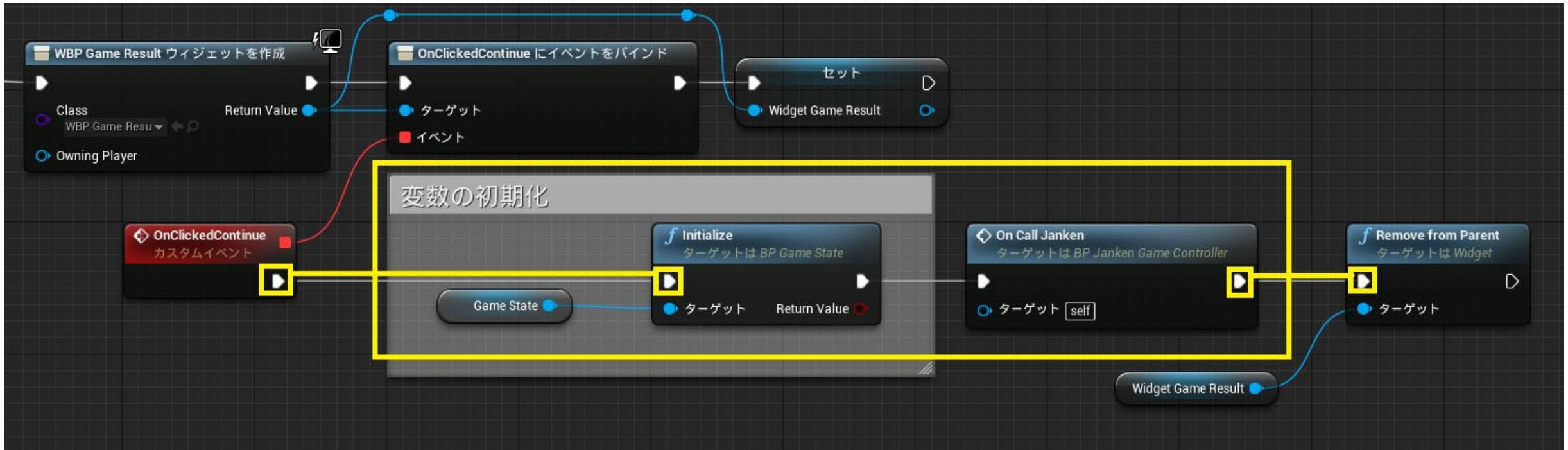
プレイして確認する



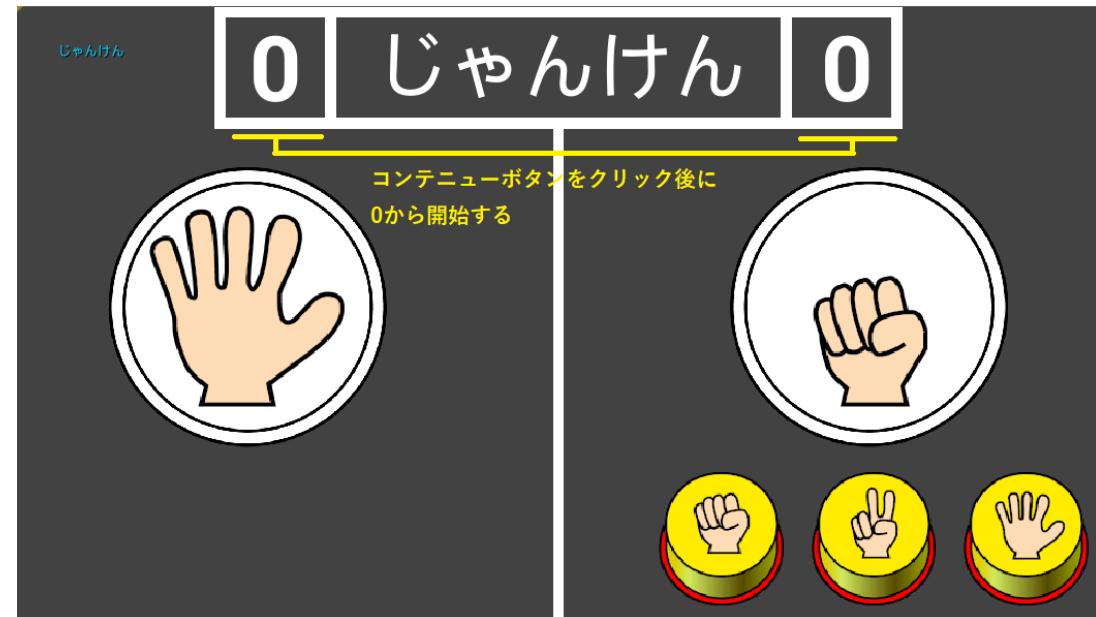
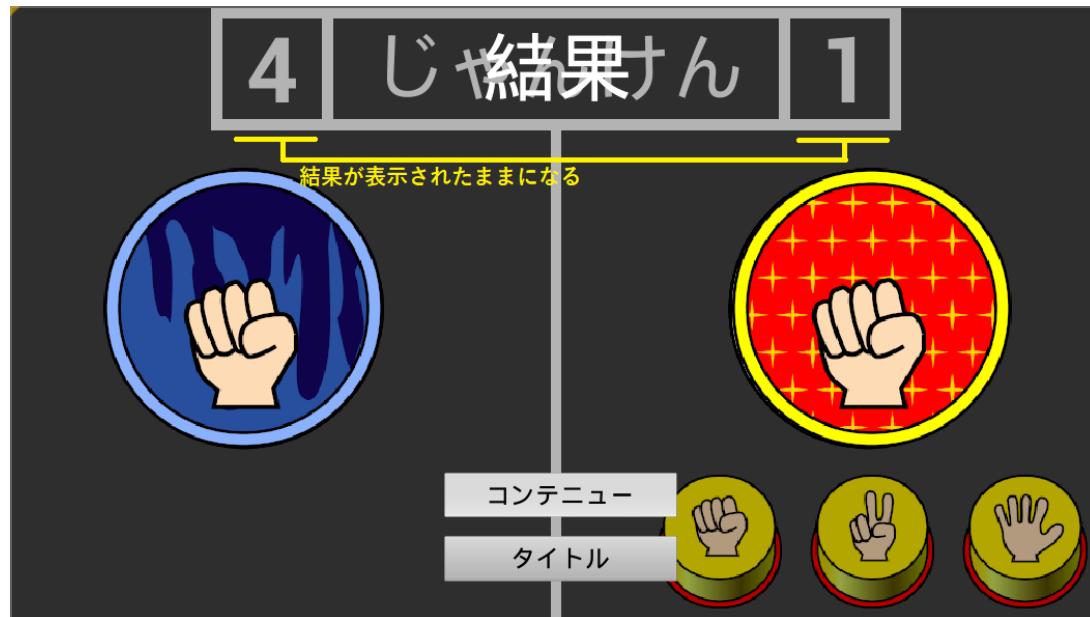
OnClickedContinue時に変数の初期化をする 1



OnClickedContinue時に変数の初期化をする 2



プレイして確認する



結果画面の文字の下を空白に設定する 1



プレイして確認する



10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

10.6.4 WBP_JankenResultに画像を設定する関数:SetJankenResultTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：SetJankenResultsを作成

10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

WBP_GameResultに勝敗の文字や背景画像を表示する



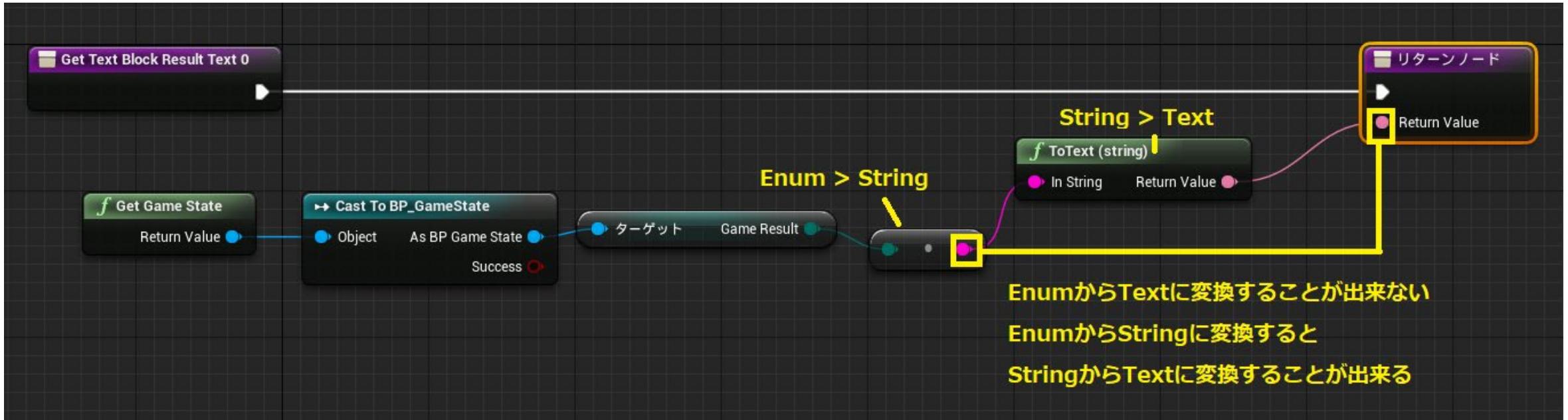
TextBlock_Resultのテキストにバインディングを作成する 1



TextBlock_Resultのテキストにバインディングを作成する 2



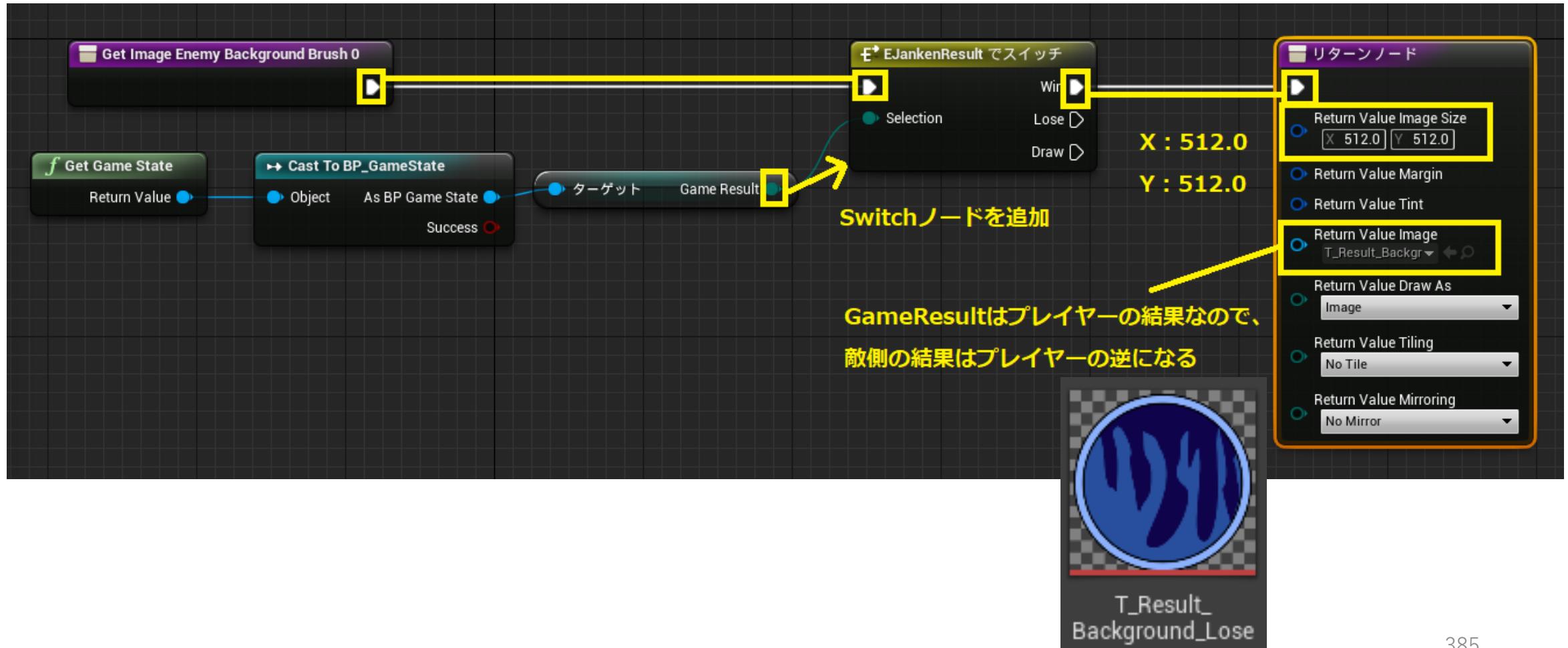
TextBlock_Resultのテキストにバインディングを作成する 3



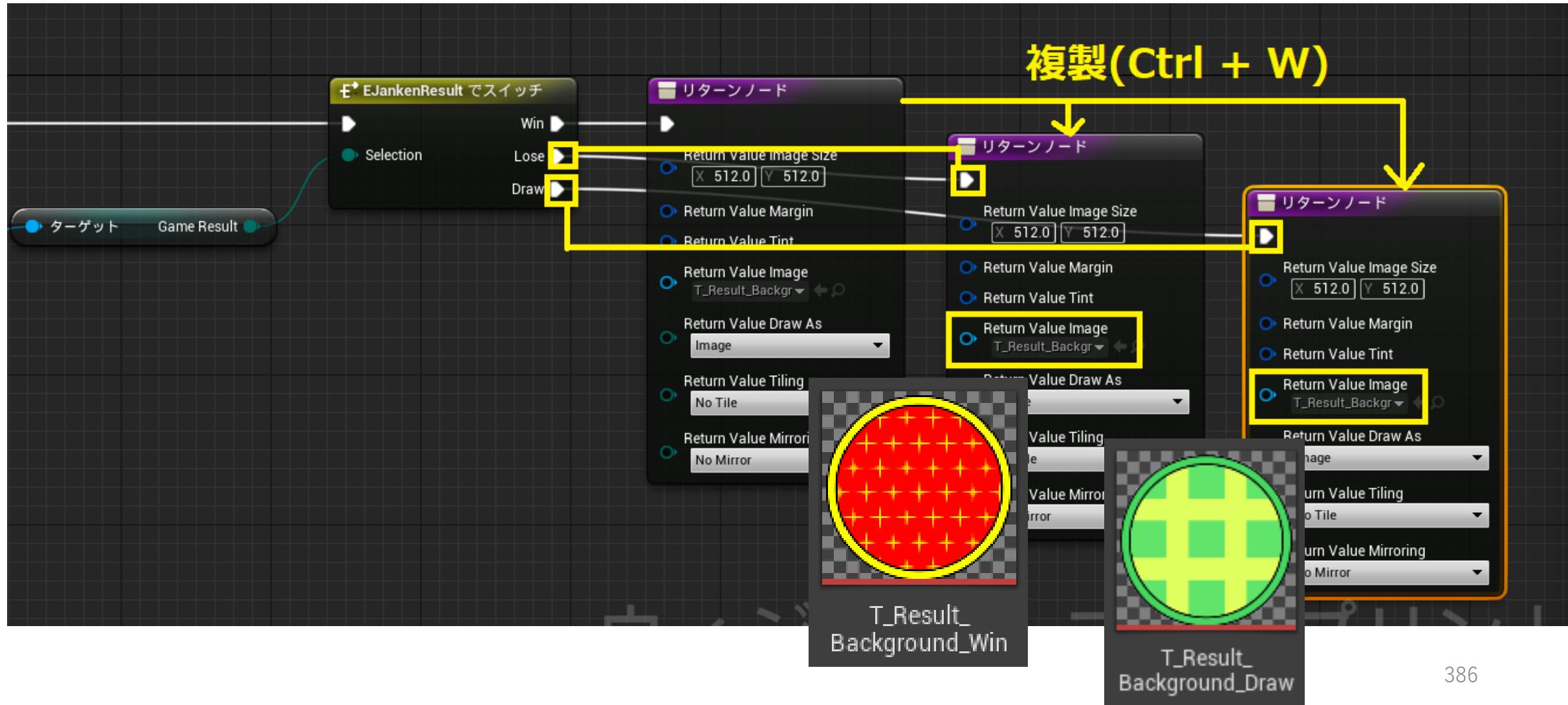
Image_EnemyBackgroundのBrushにバインディングを作成する 1



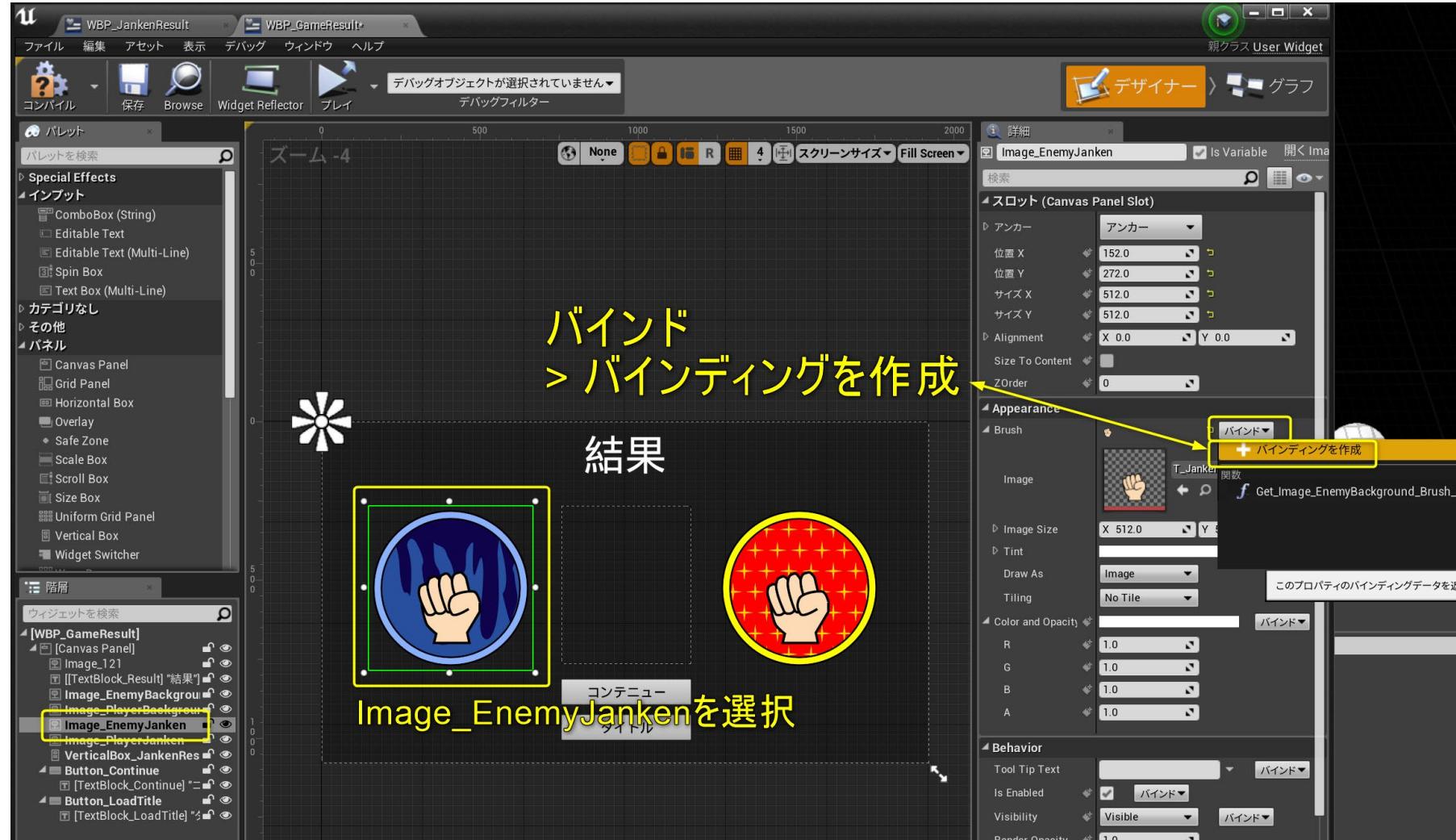
Image_EnemyBackgroundのBrushにバインディングを作成する 2



Image_EnemyBackgroundのBrushにバインディングを作成する 3



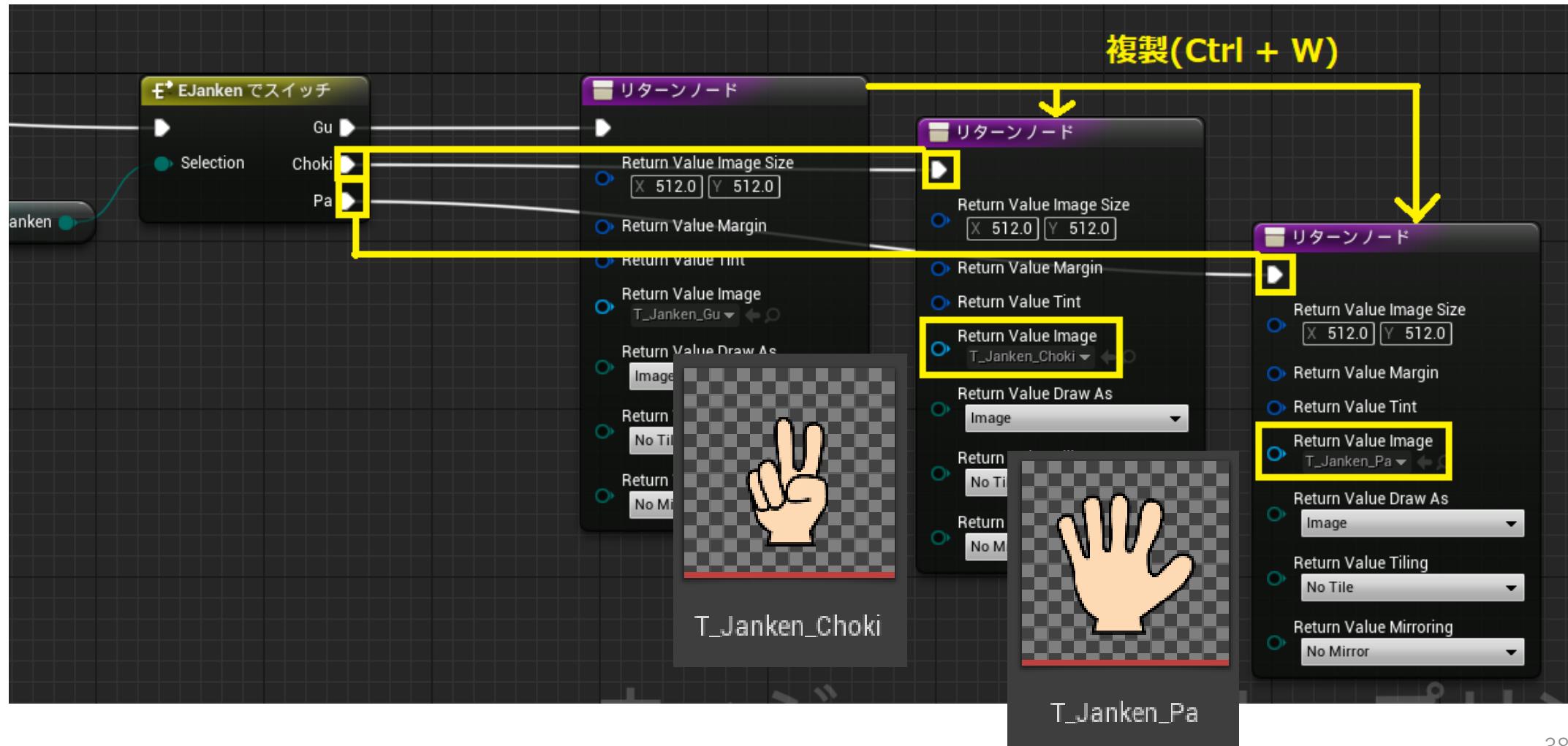
Image_EnemyJankenのBrushにバインディングを作成する 1



Image_EnemyJankenのBrushにバインディングを作成する 2



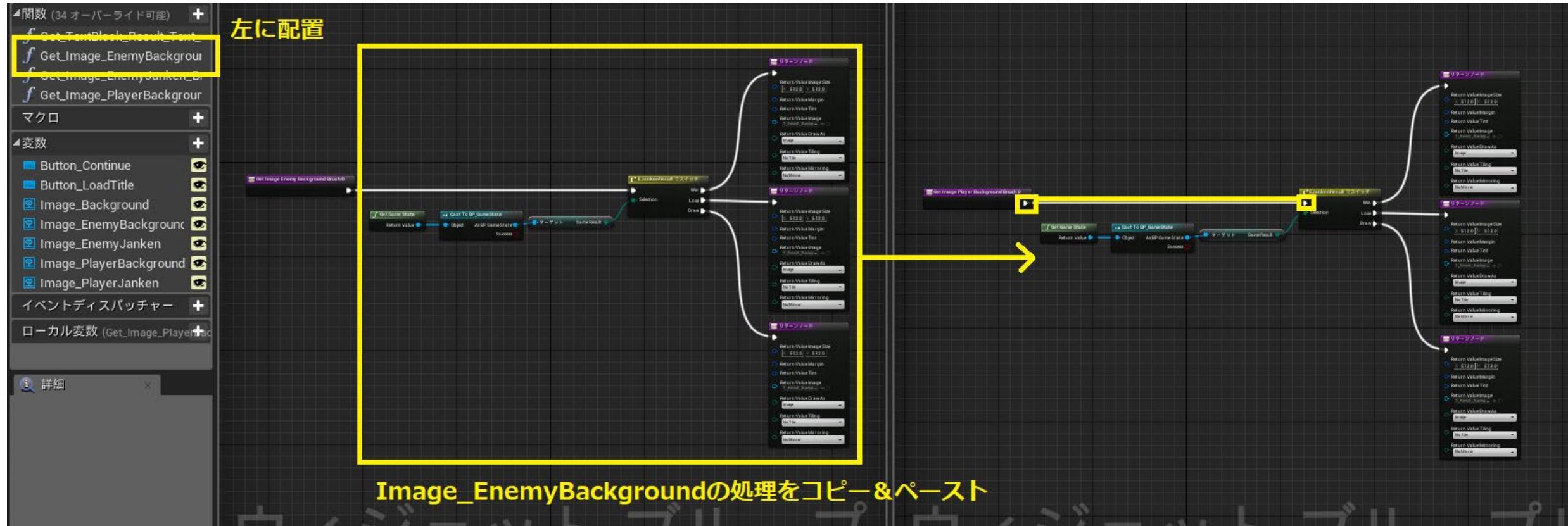
Image_EnemyJankenのBrushにバインディングを作成する 3



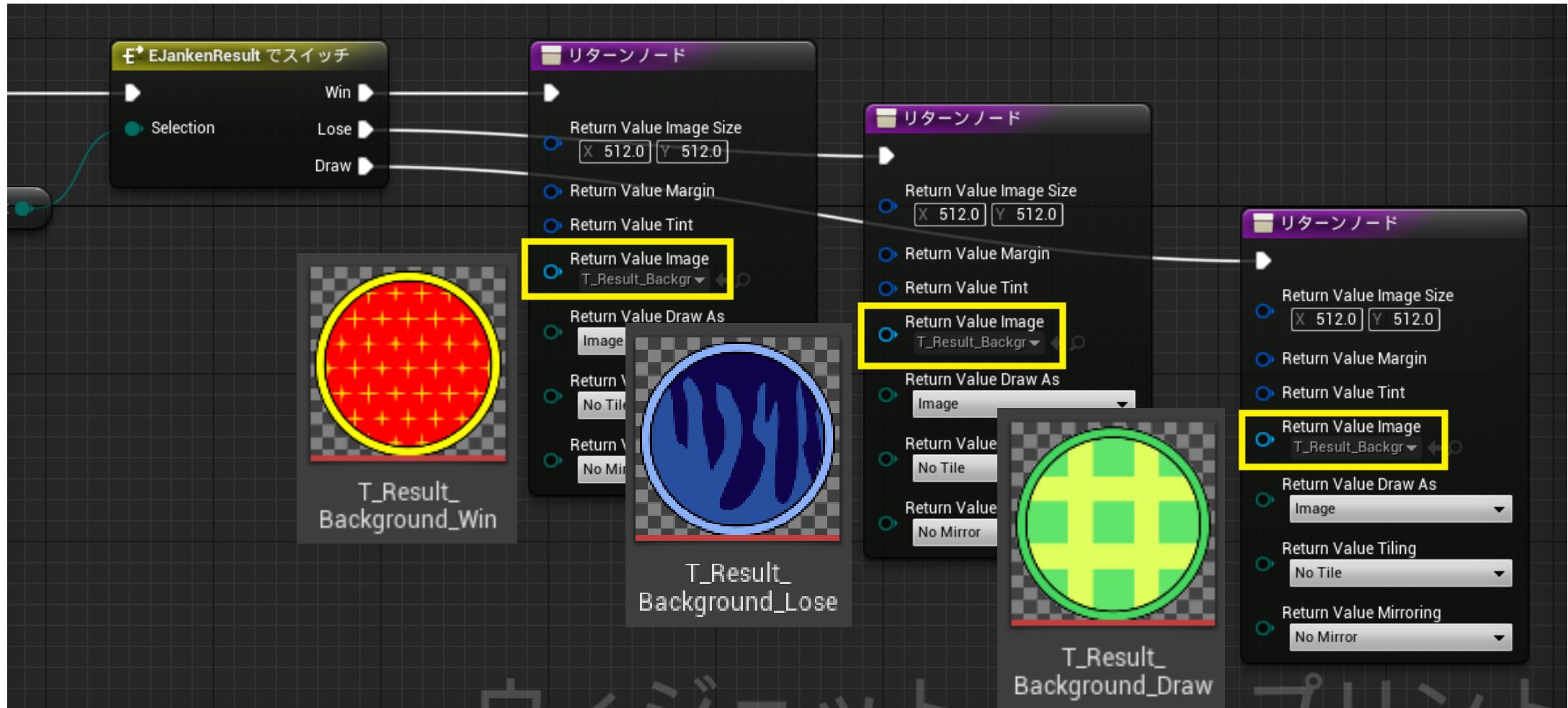
Image_PlayerBackgroundのBrushにバインディングを作成する 1



Image_PlayerBackgroundのBrushにバインディングを作成する 2



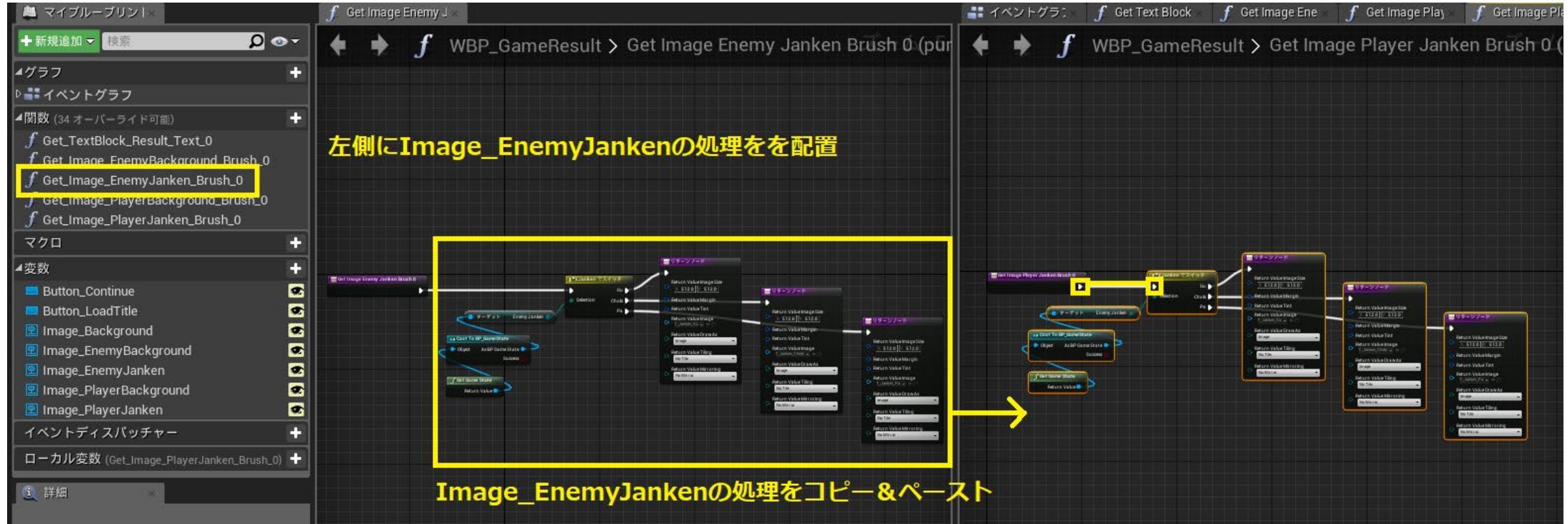
Image_PlayerBackgroundのBrushにバインディングを作成する 3



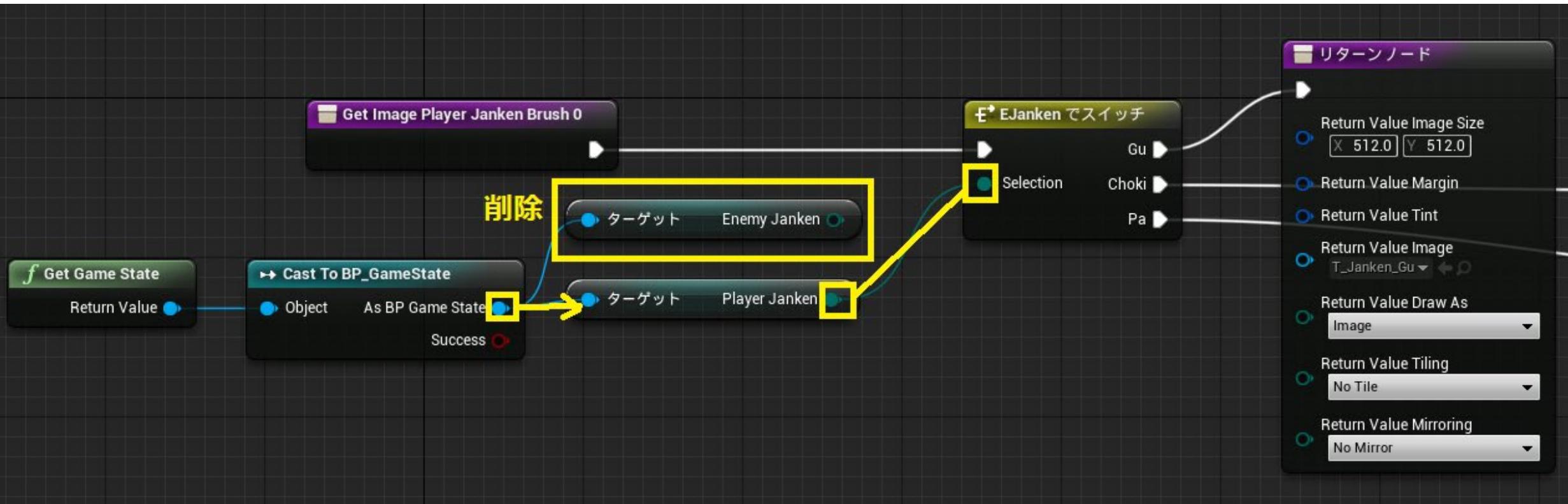
Image_Player_JankenのBrushにバインディングを作成する 1



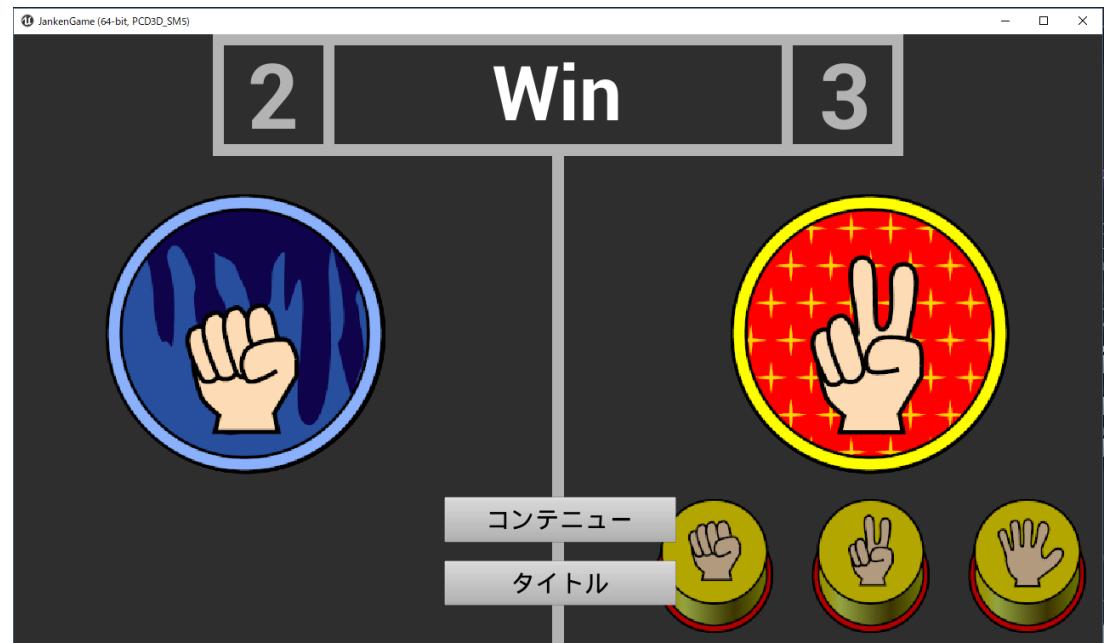
Image_Player_JankenのBrushにバインディングを作成する 2



Image_Player_JankenのBrushにバインディングを作成する 3



プレイして確認する



10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

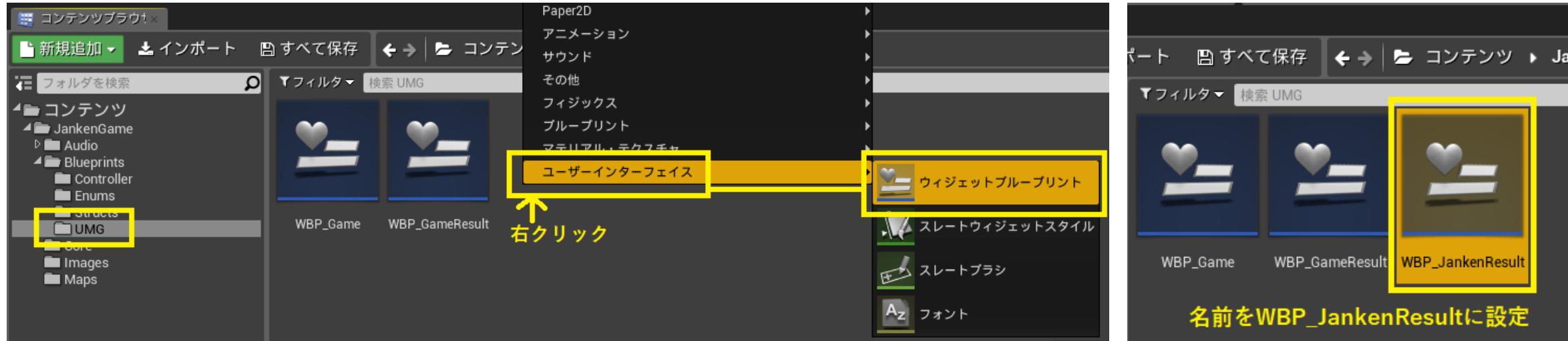
10.6.4 WBP_JankenResultに画像を設定する関数:SetJankenResultTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：SetJankenResultsを作成

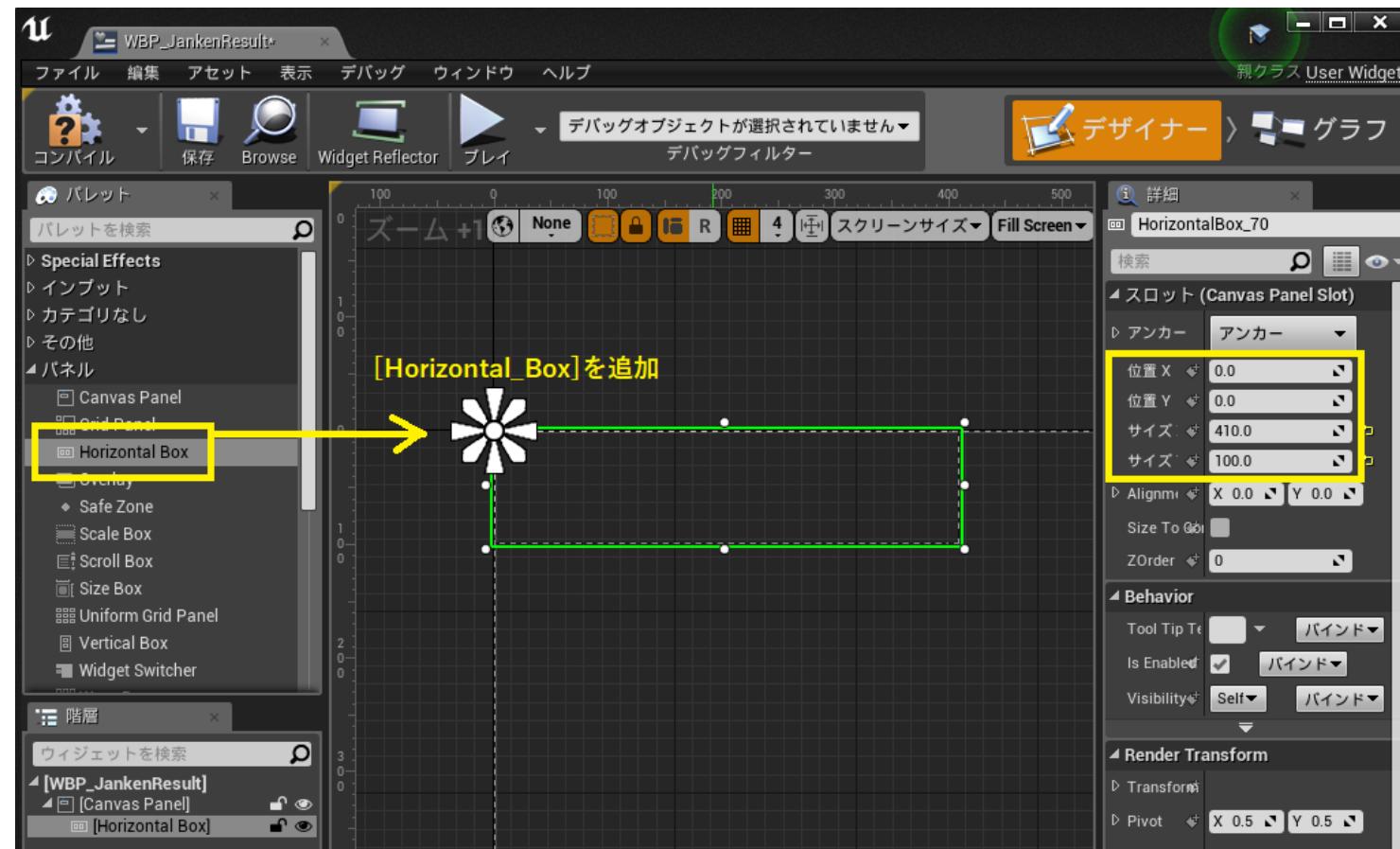
10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

ウィジェットブループリント:WBP_JankenResultを作成する



HorizontalBoxを追加する



プロパティ	値
位置X	0.0
位置Y	0.0
サイズX	410.0
サイズY	100

Image:Image_ResultEnemyJankenを追加する

The screenshot shows the Unreal Engine 4 Editor interface. On the left, the 'Palette' panel has the 'Image' item selected, highlighted with a yellow box and an arrow pointing to the workspace. The workspace contains a 'Horizontal Box' with two children: a top row containing three hand icons (Rock, Paper, Scissors) and a bottom row containing a hand icon and a red progress bar. The bottom row is labeled 'T_Janken_Gu'. The 'Details' panel on the right shows the properties for the selected 'Image_ResultEnemyJanken' asset. The 'Brush | Image' section is expanded, showing the image preview set to 'T_Janken_Gu'. Other properties include 'Size' set to 'Fill' (highlighted with a yellow box), 'Image Size' at 512x512, and 'Draw As' set to 'Image'. The 'Properties' table on the far right lists the asset's details:

プロパティ	値
変数名	Image_ResultEnemyJanken
Size	フィル
Brush Image	T_Janken_Gu

Image:Image_ResultEnemyGameを追加する

The screenshot shows the Unreal Engine 4 Editor interface. On the left, the **パレット** (Palette) panel is open, displaying various UI components. The **Image** component is selected and highlighted with a yellow box. In the center, a **Horizontal Box** is being edited. Inside the **Horizontal Box**, there is a **Image** widget with the name **Image_ResultEnemyGame**. The **Image** widget contains a hand icon and a large white 'X' symbol. To the right of the **Image** widget are two small rectangular boxes labeled **左** and **右**. A yellow arrow points from the **Image** component in the palette to the **Image** widget in the editor. Another yellow arrow points from the **[Horizontal Box]** component in the **階層** (Hierarchy) panel to the **Horizontal Box** in the editor. The **詳細** (Details) panel on the right shows the properties of the **Image** widget. The **プロパティ** (Properties) section includes:

プロパティ	値
変数名	Image_ResultEnemyGame
Size	フィル
Brush Image	T_Janken_Lose

The **Appearance** section shows the brush settings for the image, with the brush preview set to **T_Janken_Lose**.

Image:Image_ResultPlayerGameを追加する

The screenshot shows the Unreal Engine 4 Editor interface. On the left, the Content Browser displays a folder structure under [NLP_JankenResult]. A yellow box highlights the 'Image' item in the 'Image' section of the palette. In the center, the Canvas Editor shows a Horizontal Box containing three Image assets: a fist icon, a 'X' icon, and a circle icon. A yellow box highlights the 'Image_ResultPlayerGame' asset. On the right, the Details panel shows the properties for 'Image_ResultPlayerGame'. A yellow box highlights the 'Image' section under Appearance, where the brush is set to 'T_Janken_Win'. The properties table is as follows:

プロパティ	値
変数名	Image_ResultPlayerGame
Size	フィル
Brush Image	T_Janken_Win

Image:Image_ResultPlayerJankenを追加する

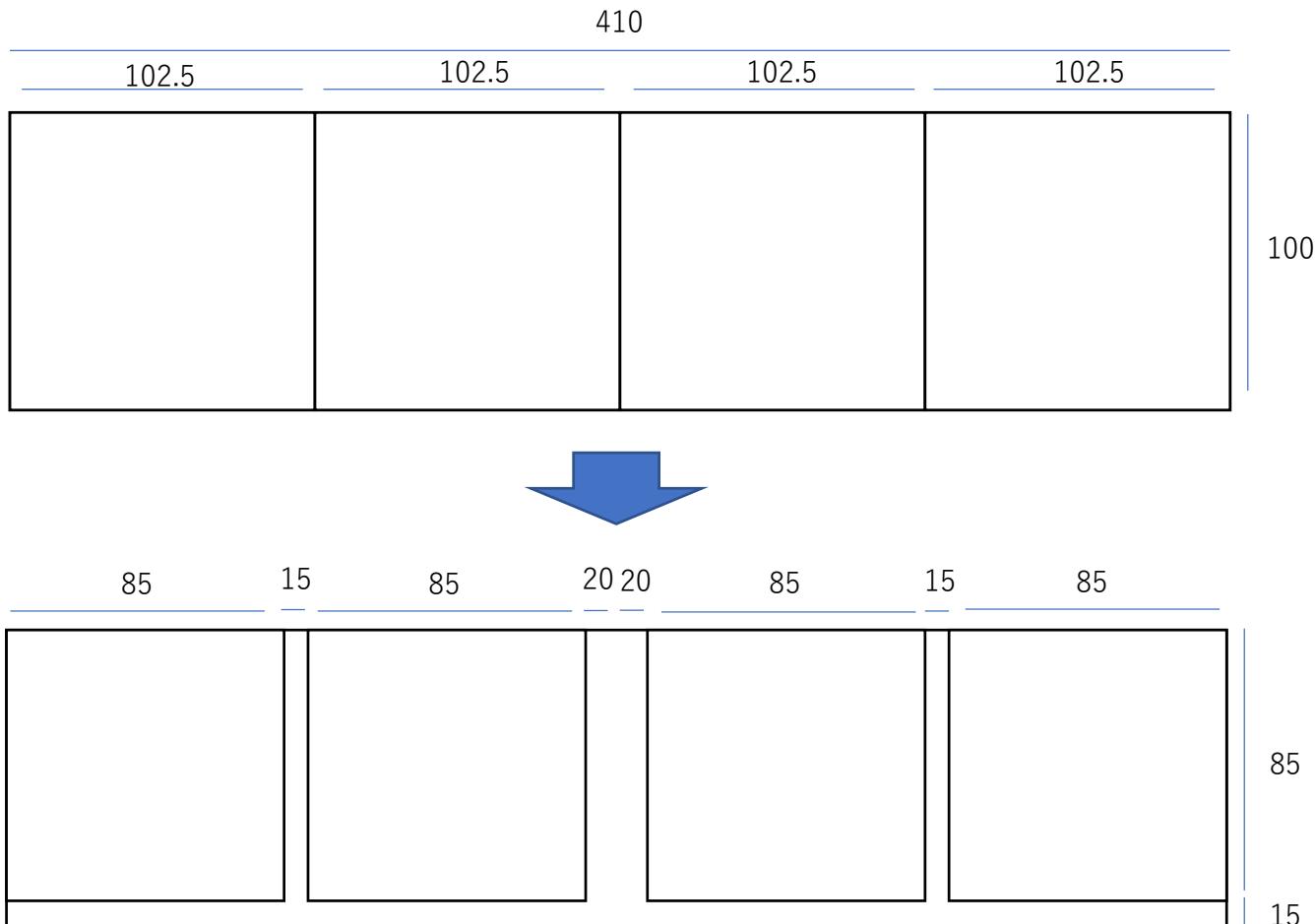
The screenshot shows the Unreal Engine 4 Editor interface with the following components visible:

- Palette (Left):** Shows various UI elements like Vertical Box, Widget Switcher, Wrap Box, etc. The "Image" item is highlighted with a yellow box.
- Canvas Panel (Center):** A "Horizontal Box" containing three items: a fist icon, a large "X" icon, and an open hand icon. Labels "左" (Left) and "右" (Right) are placed next to the middle and right icons respectively. A yellow box highlights the "Image" item in the palette.
- Details View (Top Right):** Shows the properties for the selected "Image" component.
 - Image_ResultPlayerJanken:** The component name.
 - Slot (Horizontal Box Slot):** Padding is set to 0.0, Size is set to "自動" (Automatic) with a value of 1.0.
 - Appearance:** Brush is set to "T_Janken_Pa".
 - Image Size:** X: 512.0, Y: 512.0.
 - Behavior:** Tool Tip Text is empty, Is Enabled is checked.A yellow box highlights the "Image" component in the details view.
- Properties View (Bottom Right):** Displays the properties for the "Image" component:

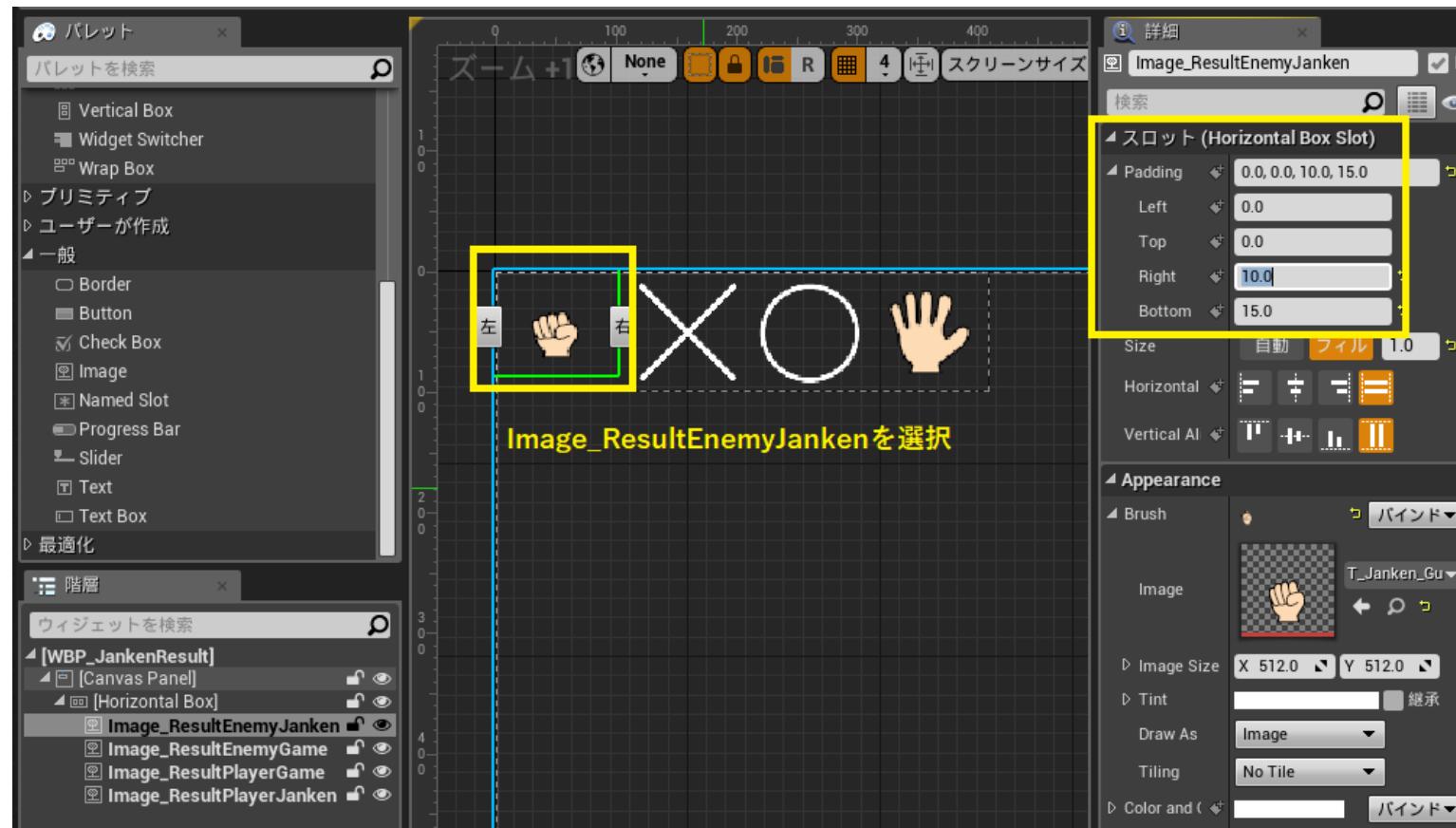
プロパティ	値
変数名	Image_ResultPlayerJanken
Size	フィル
Brush Image	T_Janken_Pa

[Image]をHorizontal_Boxに追加 (Add [Image] to Horizontal_Box) is displayed at the bottom left of the canvas.

Paddingを変更する



Image_ResultEnemyJankenのPaddingを変更する



プロパティ	値
Padding Left	0.0
Padding Top	0.0
Padding Right	15.0
Padding Bottom	15.0

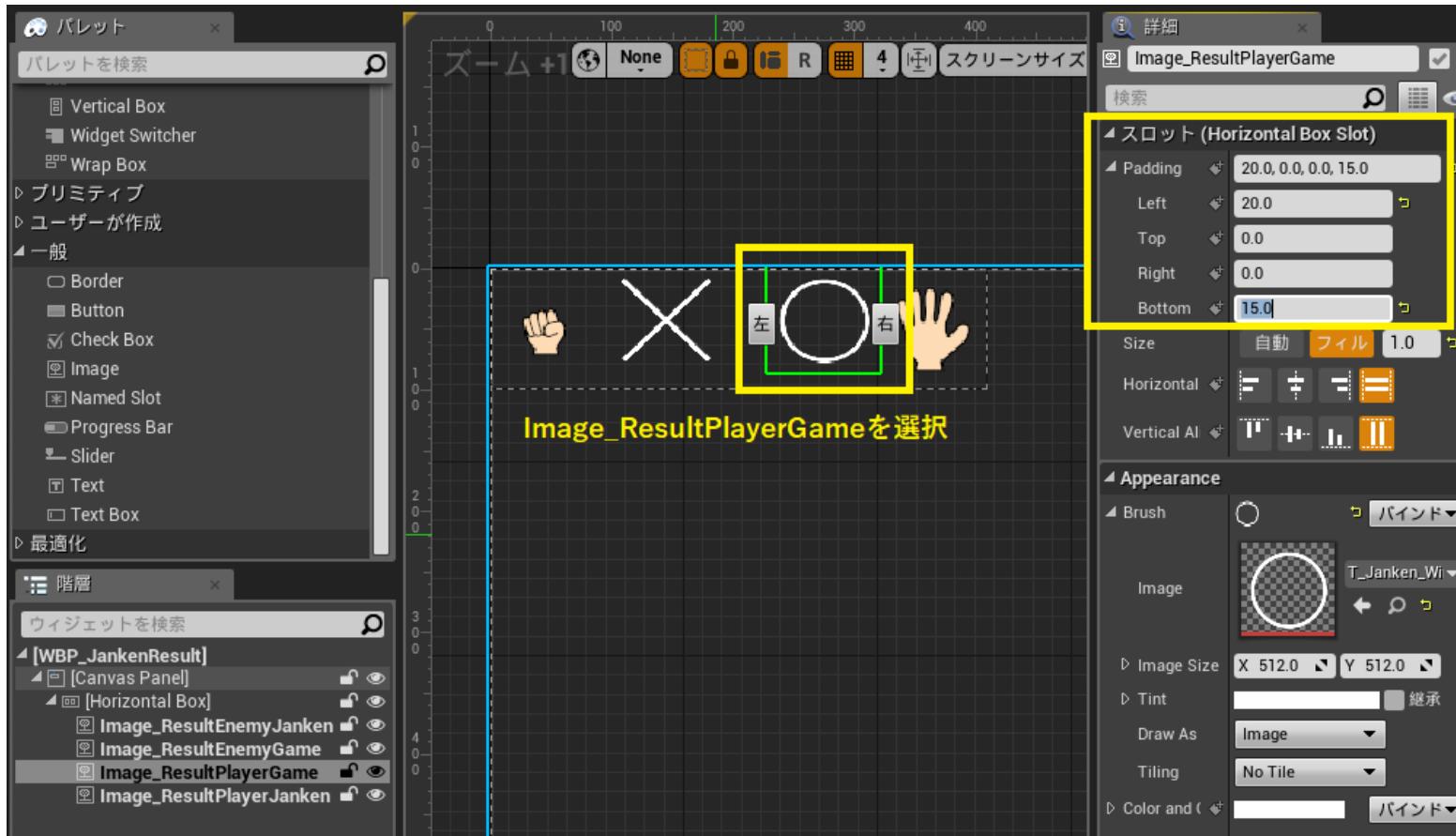
Image_ResultEnemyGameのPaddingを変更する

The screenshot shows the Unreal Engine 4 Editor interface with the following components:

- Palette (Left):** Shows various UI widget types like Vertical Box, Widget Switcher, Wrap Box, etc.
- Canvas Panel (Center):** Displays a horizontal box containing three images: a fist icon, a hand icon with a cross, and a circle icon. A green rectangular selection box highlights the middle image. A yellow box highlights the "Horizontal Box Slot" in the Details panel.
- Details Panel (Right):** Shows properties for the selected "Image_ResultEnemyGame" component.
 - Padding:** Set to 0.0, 0.0, 20.0, 15.0 (Left, Top, Right, Bottom).
 - Appearance:** Brush is set to "T_Janken_Lo".
 - Image:** T_Janken_Lo
 - Image Size:** X: 512.0, Y: 512.0
 - Tint:** Gray
 - Draw As:** Image
 - Tiling:** No Tile
- Layer View (Bottom Left):** Shows the current layer structure, including [WBP_JankenResult] and its children: [Canvas Panel], [Horizontal Box], and the selected [Image_ResultEnemyGame].
- Table (Right):** Summarizes the padding values.

プロパティ	値
Padding Left	0.0
Padding Top	0.0
Padding Right	20.0
Padding Bottom	15.0

Image_ResultPlayerGameのPaddingを変更する



プロパティ	値
Padding Left	20.0
Padding Top	0.0
Padding Right	0.0
Padding Bottom	15.0

Image_ResultPlayerJanKenのPaddingを変更する

The screenshot shows the Unreal Engine 4 Editor interface with several open panels:

- パレット (Palette):** Shows various UI components like Vertical Box, Widget Switcher, Wrap Box, etc.
- 階層 (Hierarchy):** Shows the scene structure: [WBP_JankenResult] -> [Canvas Panel] -> [Horizontal Box] -> [Image_ResultEnemyJanken], [Image_ResultEnemyGame], [Image_ResultPlayerGame], and [Image_ResultPlayerJanKen].
- 詳細 (Details):** The panel for the selected component, **Image_ResultPlayerJanKen**, is highlighted with a yellow box. It displays the following properties:

プロパティ (Property)	値 (Value)
Padding Left	15.0
Padding Top	0.0
Padding Right	0.0
Padding Bottom	15.0

The canvas view shows a horizontal box containing four images: a fist icon, an 'X' icon, a circle icon, and a hand icon. The hand icon is highlighted with a green selection box and has a yellow bounding box around its bounding box in the details panel.

10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

10.6.4 WBP_JankenResultに画像を設定する関数:SetJankenResultTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：SetJankenResultsを作成

10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

WBP_JankenResultに関する関数:GetJankenTextureを追加する 1



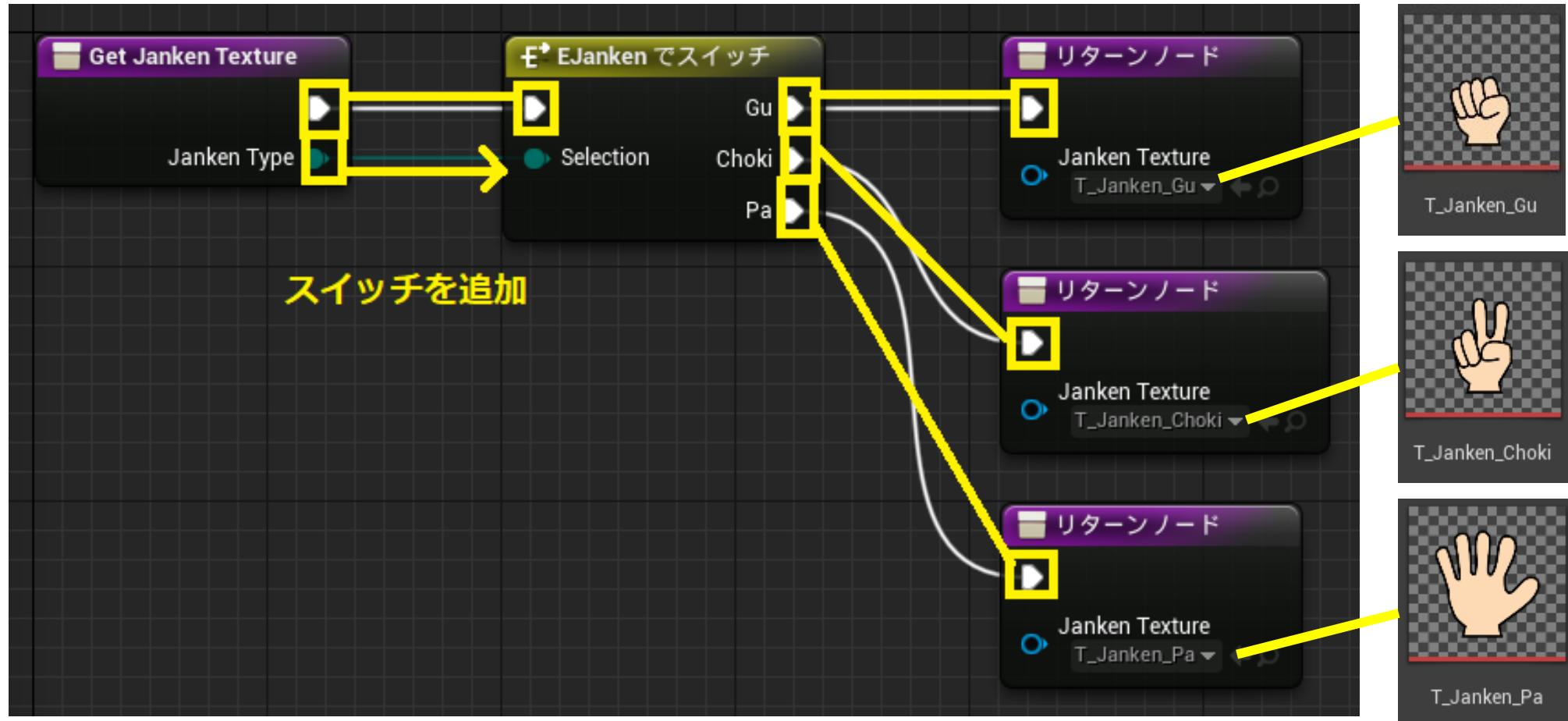
関数名	アクセス指定子	Input/Output	パラメータ名	型
GetJankenTexture	プライベート	Input	JankenType	EJanken
		Output	JankenTexture	Texture2D (Object Reference)

説明

じゃんけんの画像を取得する

ウェイジ

WBP_JankenResultに関する関数:GetJankenTextureを追加する 2



10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

10.6.4 WBP_JankenResultに画像を設定する関数:SetJankenResultTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：SetJankenResultsを作成

10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

WBP_JankenResultに関する関数:GetResultGameTextureを追加する 1

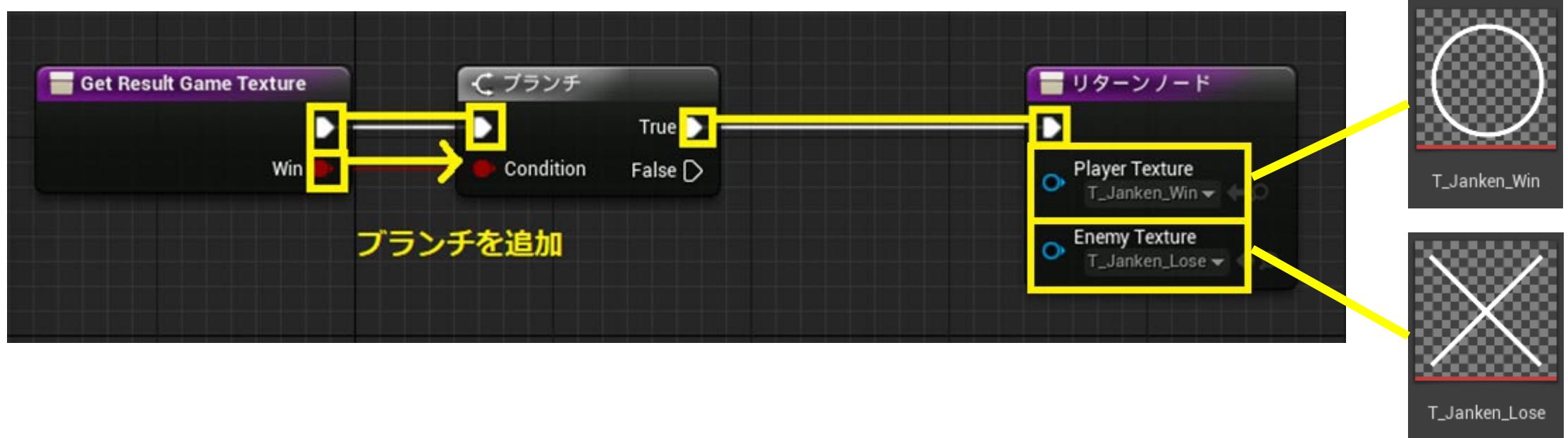


関数名	アクセス指定子	Input/Output	パラメータ名	型
GetResultGameTexture	プライベート	Input	Win	Boolean
		Output	PlayerTexture	Texture2D (Object Reference)
		Output	EnemyTexture	Texture2D (Object Reference)

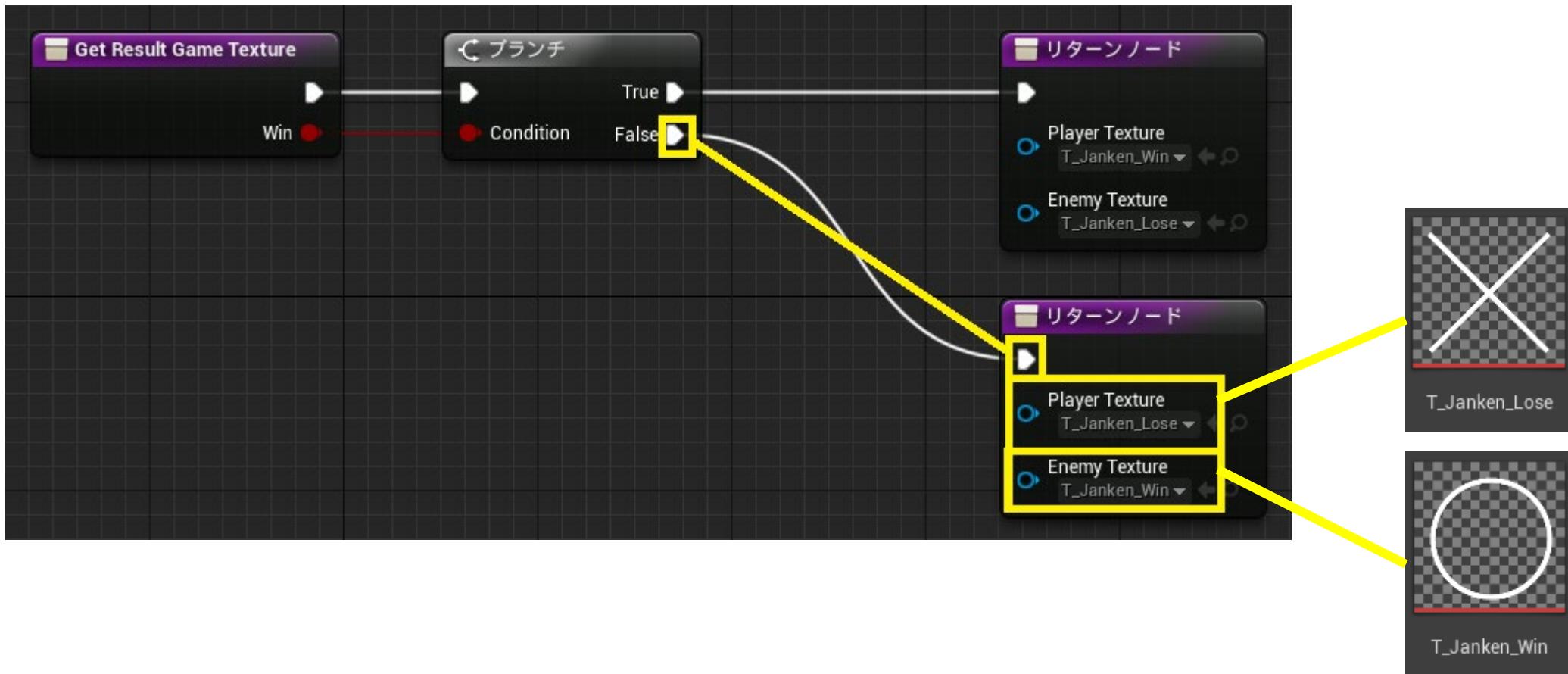
説明

じゃんけんの勝敗画像を取得する

WBP_JankenResultに関数:GetResultGameTextureを追加する 2



WBP_JankenResultに関数:GetResultGameTextureを追加する 3



10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

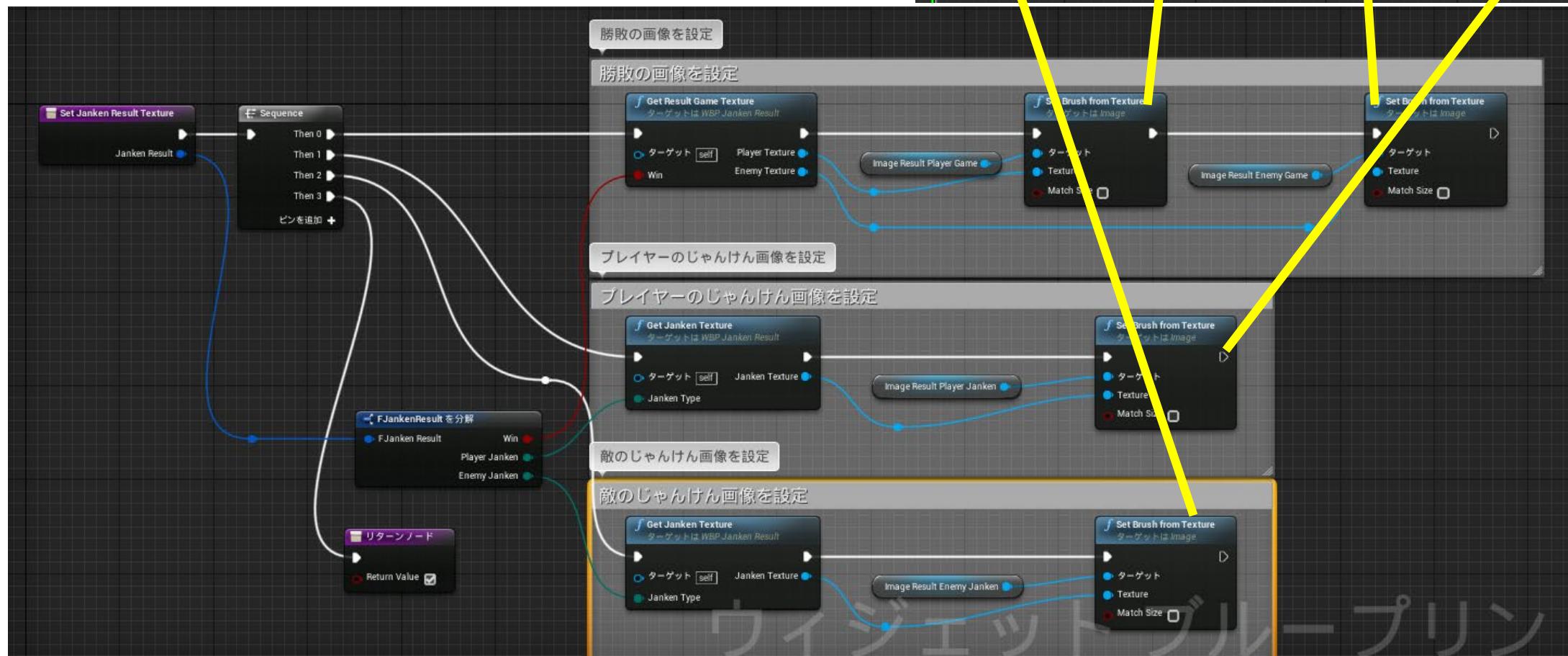
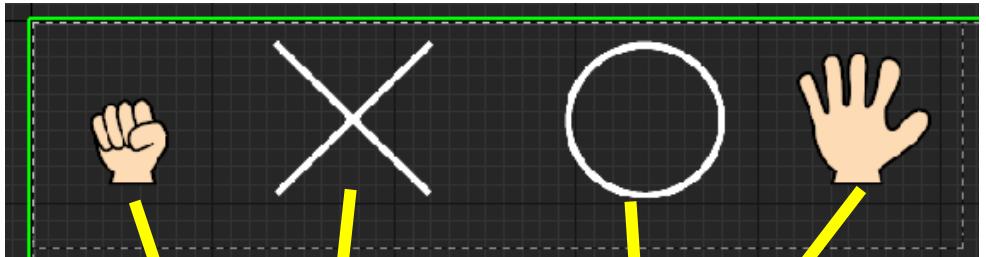
10.6.4 WBP_JankenResultに画像を設定する関数:SetJankenResultTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：SetJankenResultsを作成

10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

WBP_JankenResultに画像を設定する 関数:SetJankenResultTextureを作成する



WBP_JankenResultに画像を設定する 関数:SetJankenResultTextureを作成する 1

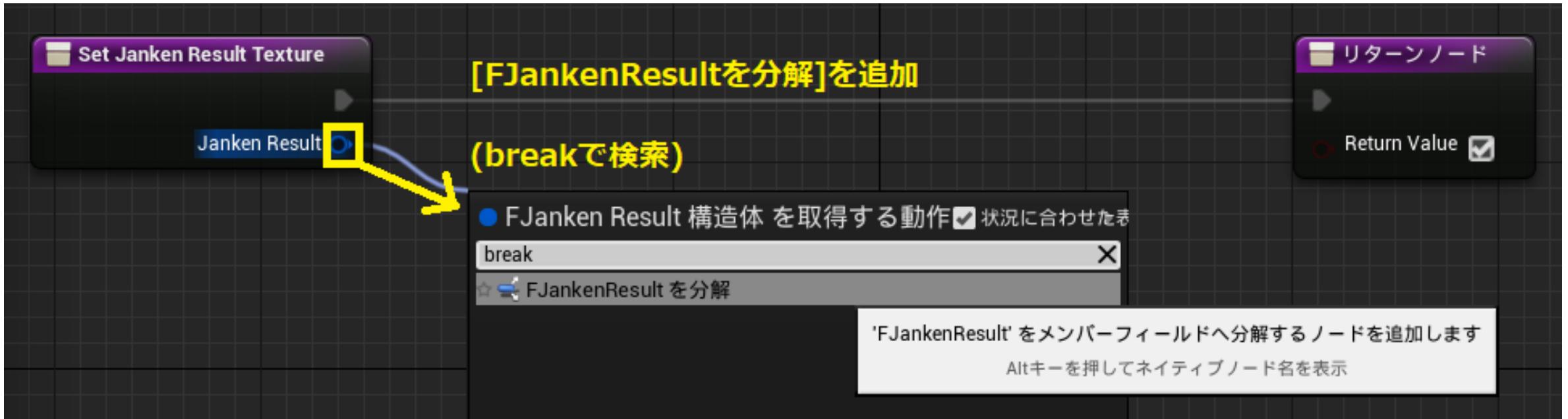
The screenshot shows the Unreal Engine Blueprint Editor interface. On the left, the Function Library panel lists several functions, with 'f SetJankenResultTexture' highlighted and a yellow box around it. Below it, the Graph panel shows a new node being created with the title '新規ノードを作成' (Create New Node). The node has one input port 'Janken Result' and one output port 'Return Value'. A yellow box highlights the 'Return Value' port. To the right, a preview window titled 'WBP_Janken Result' displays a dark background with a white heart icon and two white rectangular bars below it. At the bottom, there is a detailed table about the function and a descriptive text box.

関数名	アクセス指定子	Input/Output	パラメータ名	型
SetJankenResultTexture	パブリック	Input	JankenResult	FJankenResult
		Output	ReturnValue	Boolean

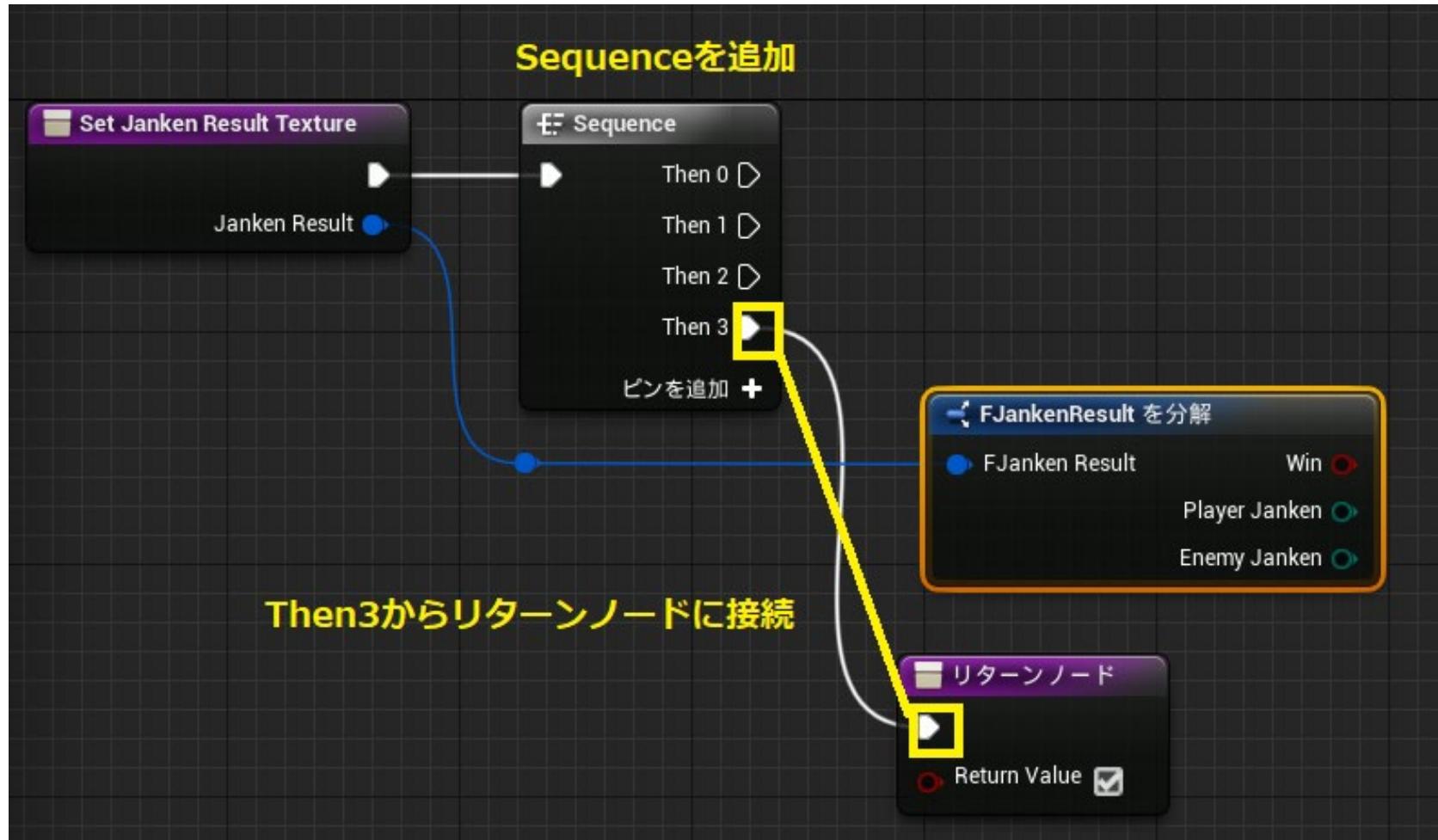
説明 FJankenResultから画像を設定する

418

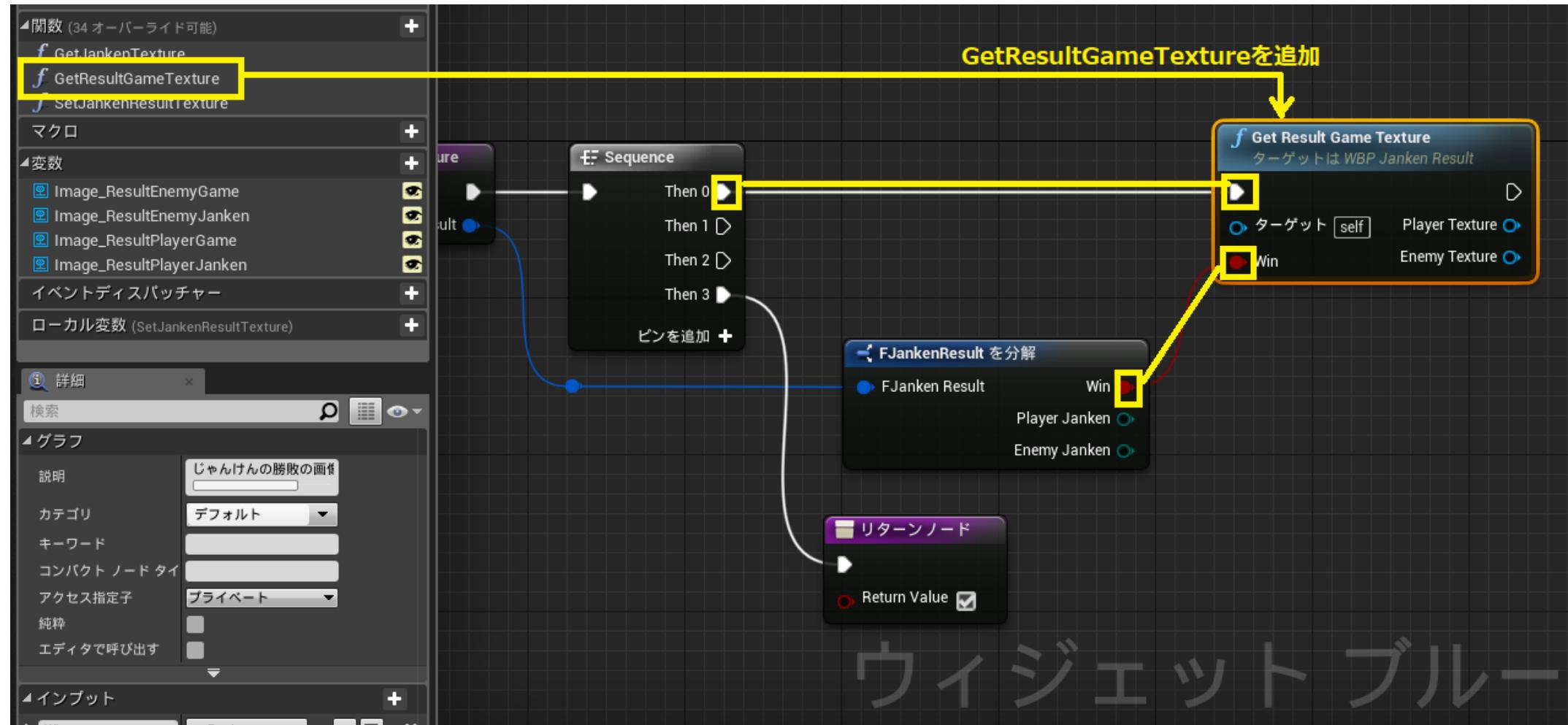
WBP_JankenResultに画像を設定する 関数:SetJankenResultTextureを作成する 2



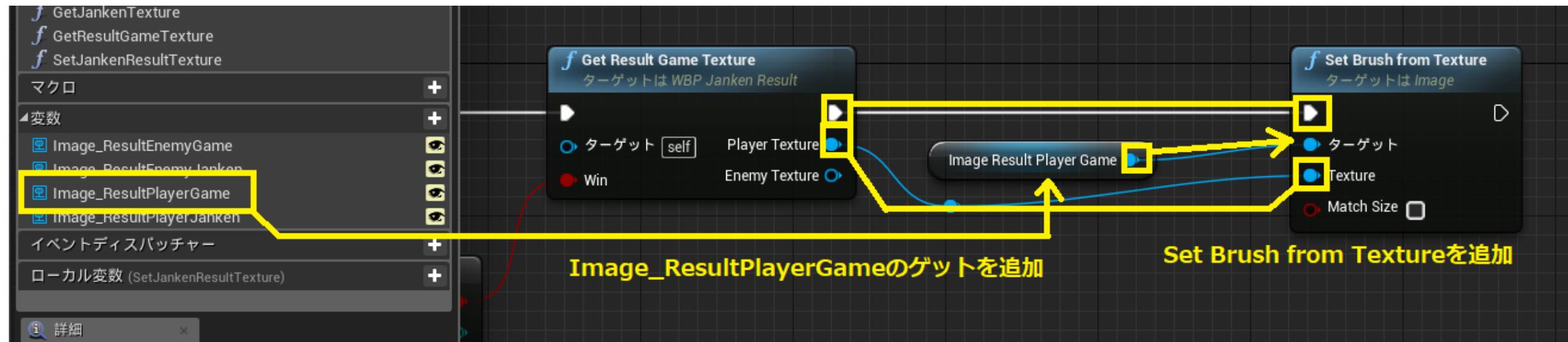
WBP_JankenResultに画像を設定する 関数:SetJankenResultTextureを作成する 3



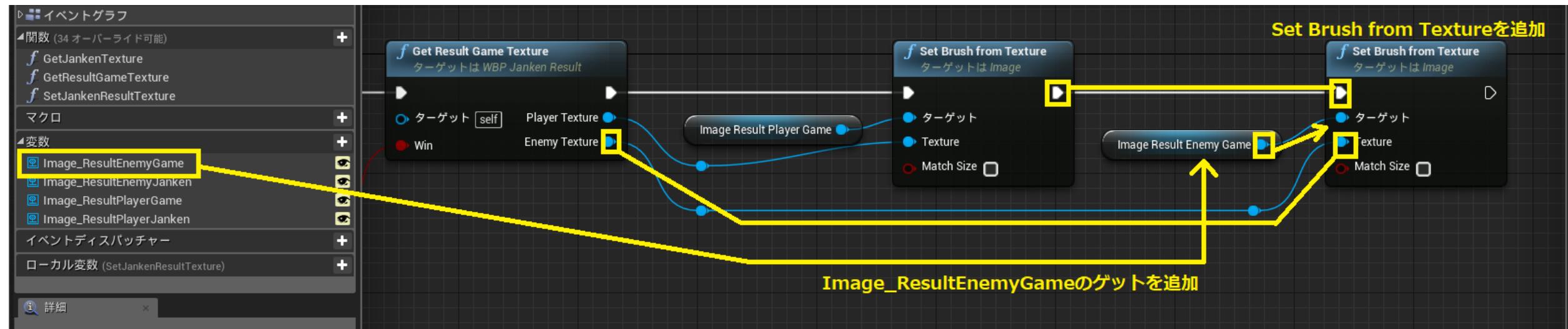
WBP_JankenResultに画像を設定する 関数:SetJankenResultTextureを作成する 4



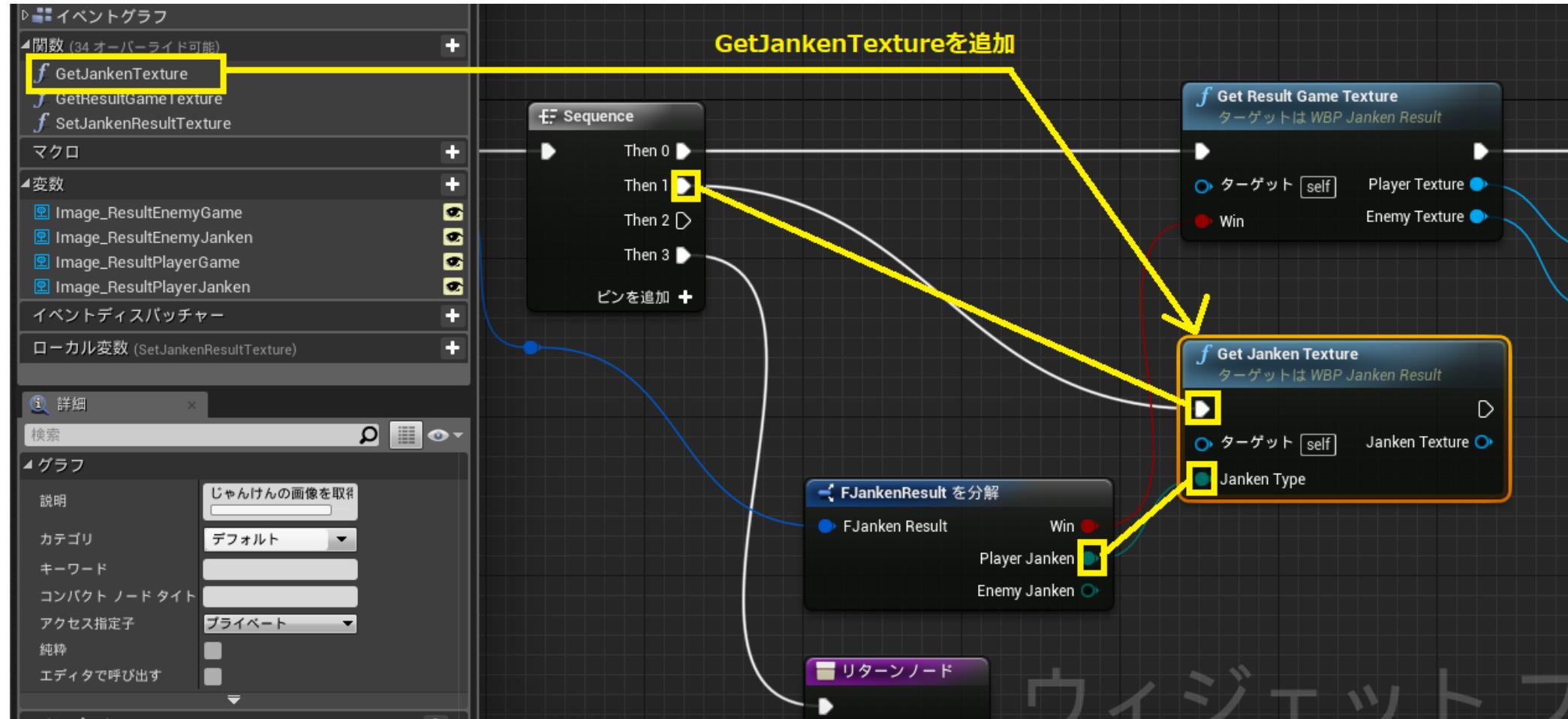
WBP_JankenResultに画像を設定する 関数:SetJankenResultTextureを作成する 5



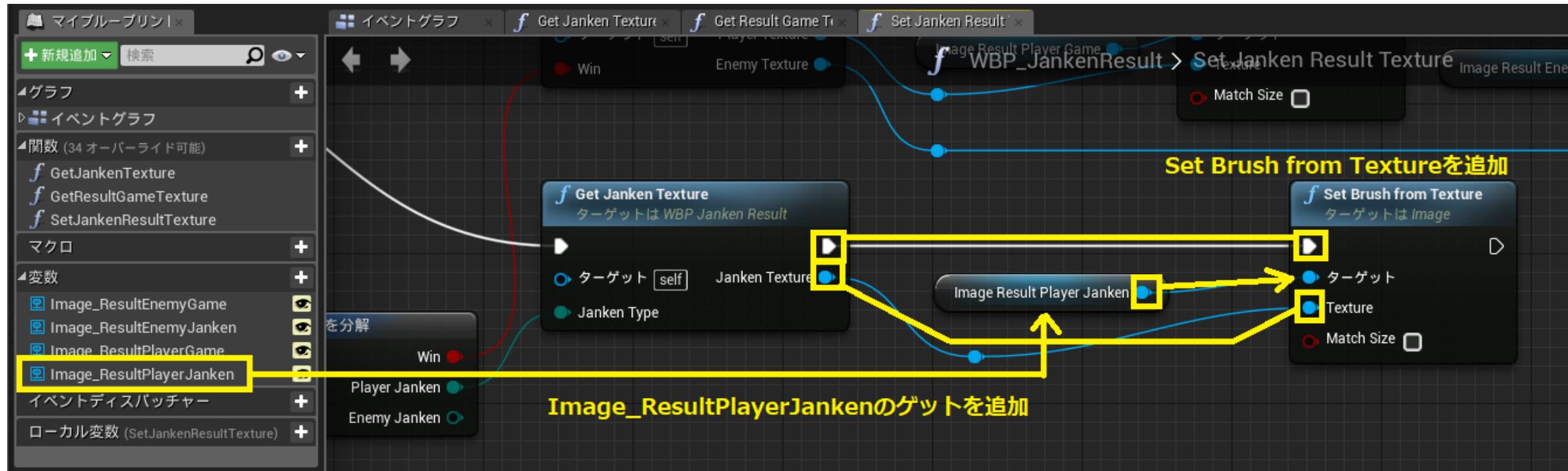
WBP_JankenResultに画像を設定する 関数:SetJankenResultTextureを作成する 6



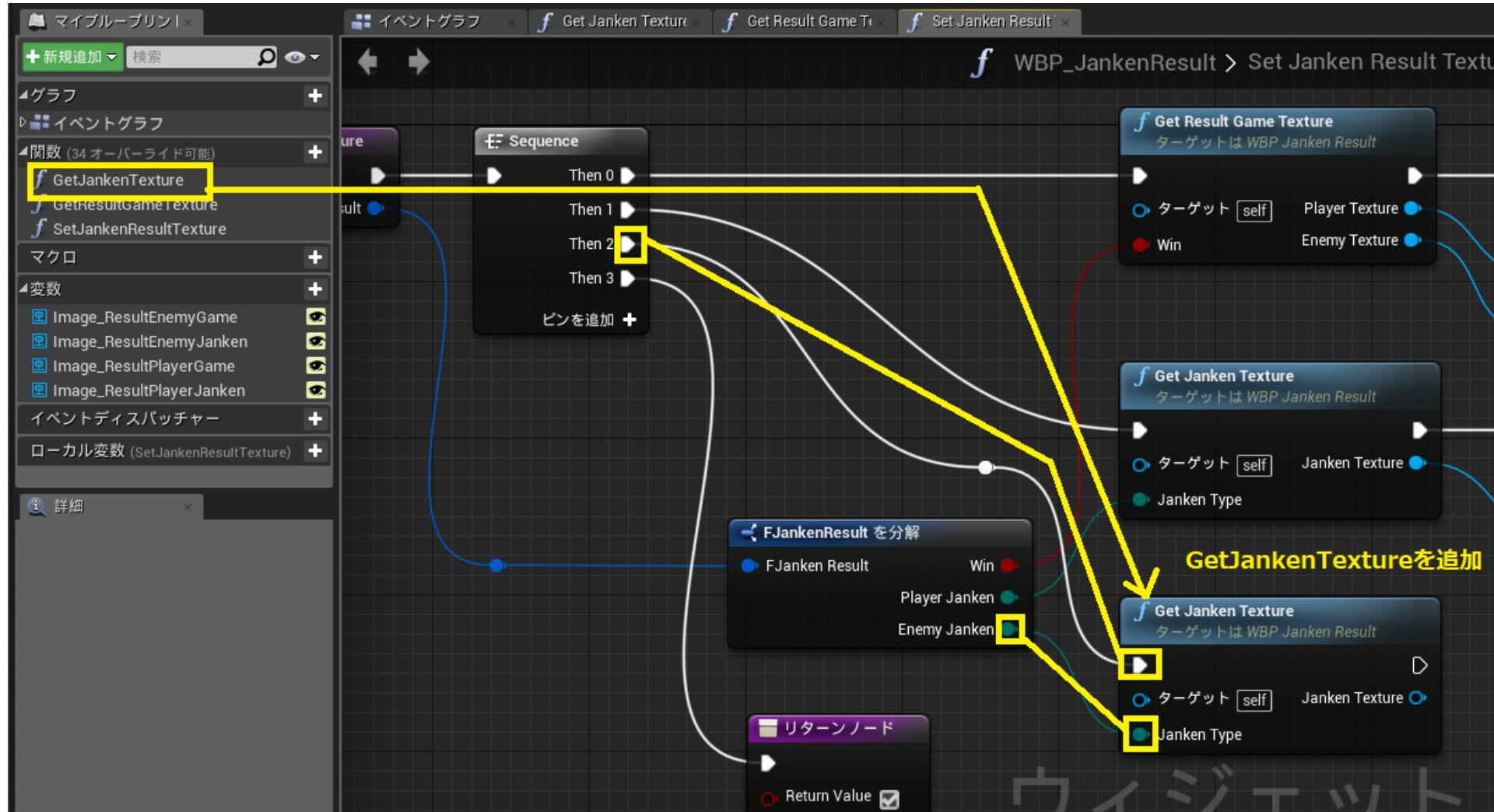
WBP_JankenResultに画像を設定する 関数:SetJankenResultTextureを作成する 7



WBP_JankenResultに画像を設定する 関数:SetJankenResultTextureを作成する 8

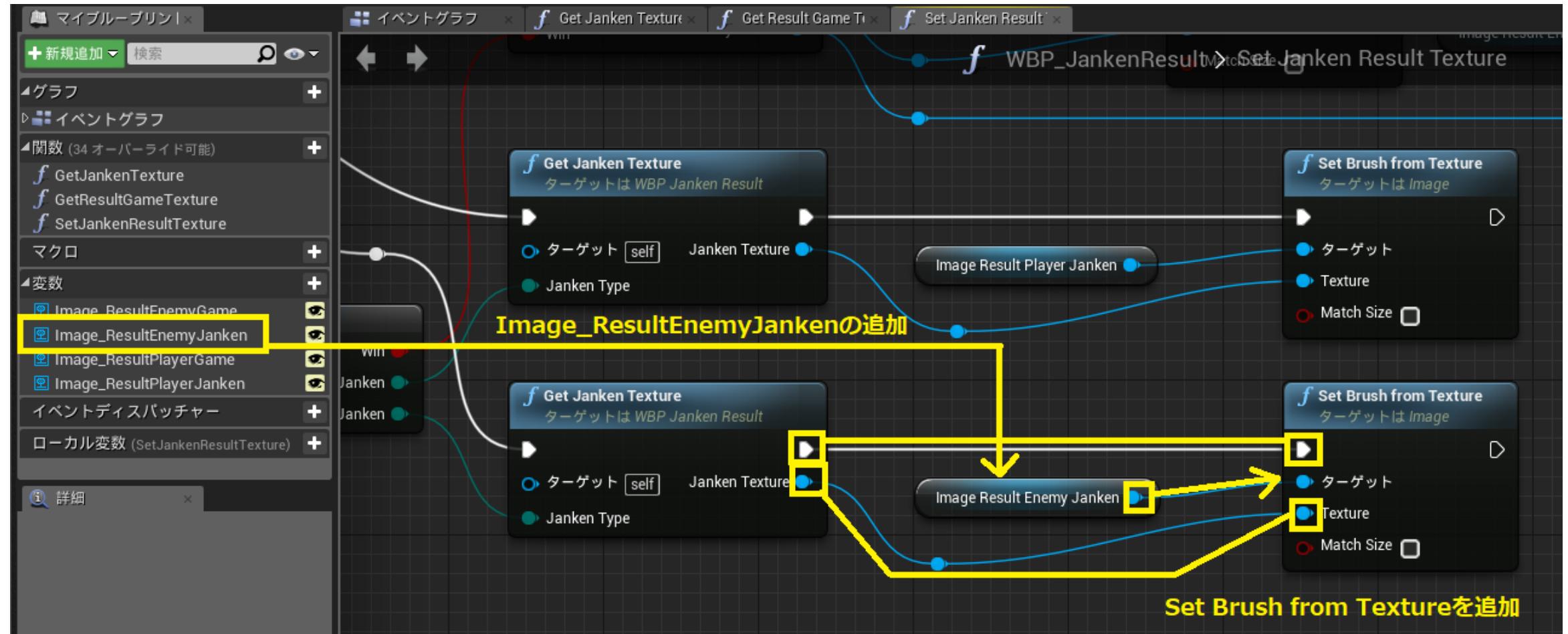


WBP_JankenResultに画像を設定する 関数:SetJankenResultTextureを作成する 9

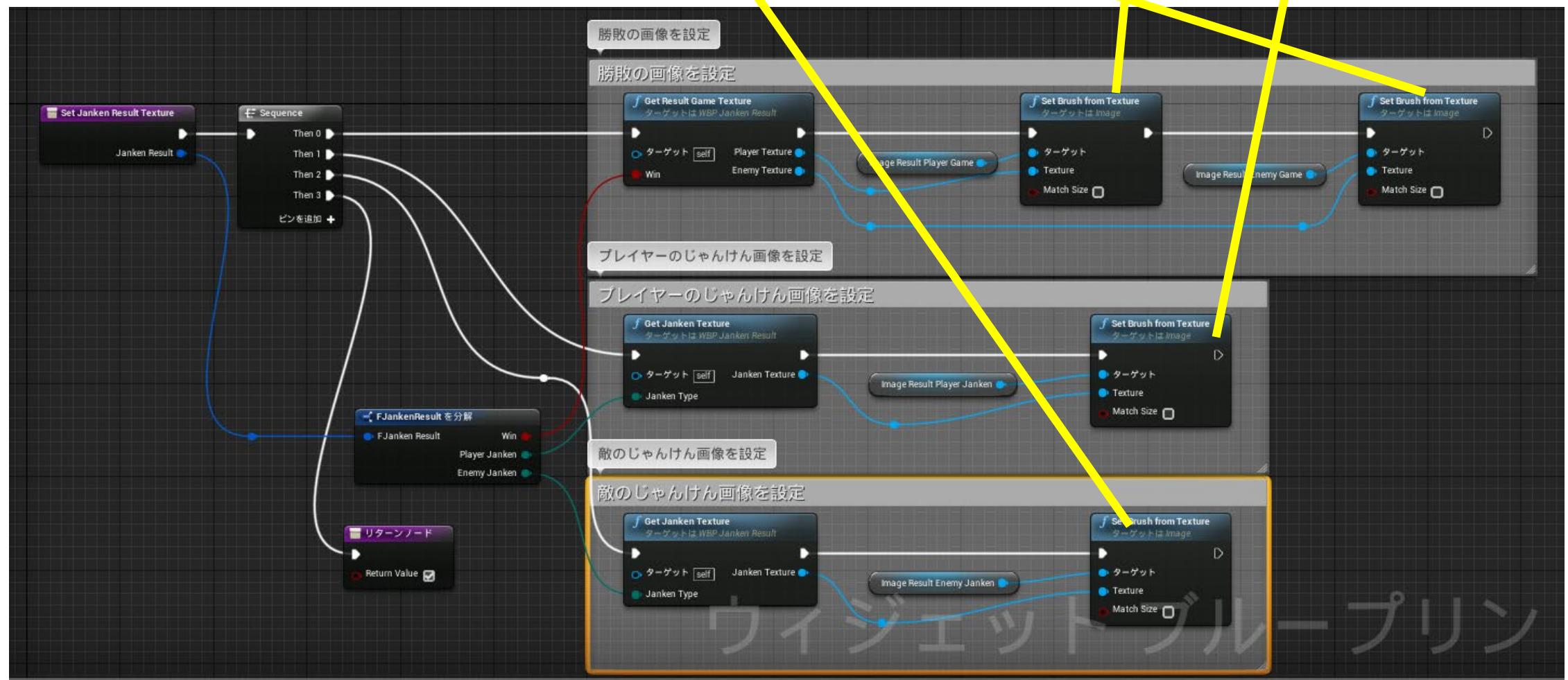


ウィジット

WBP_JankenResultに画像を設定する 関数:SetJankenResultTextureを作成する 10



完成図



コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

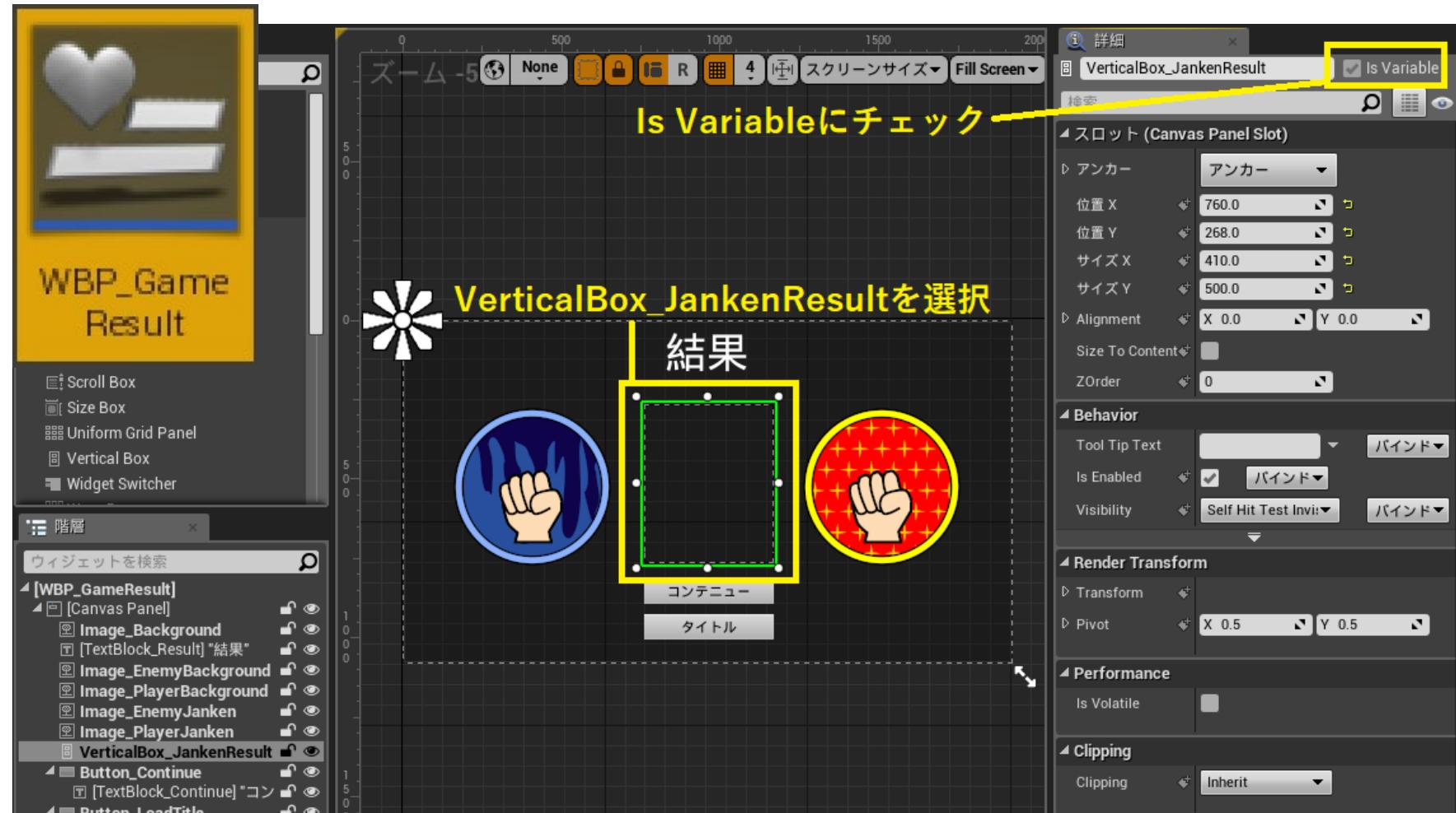
10.6.4 WBP_JankenResultに画像を設定する関数:SetJankenResultTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：SetJankenResultsを作成

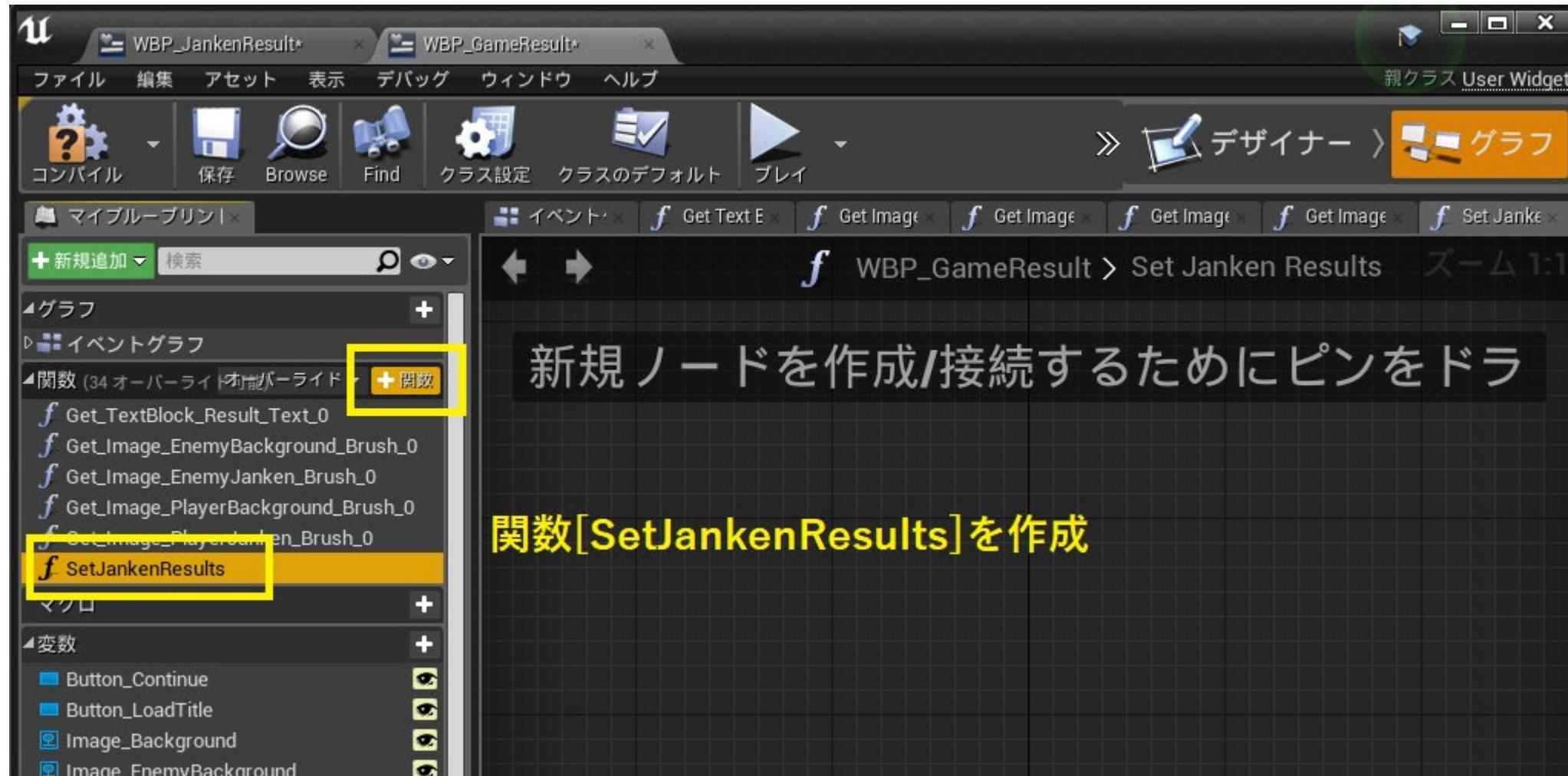
10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

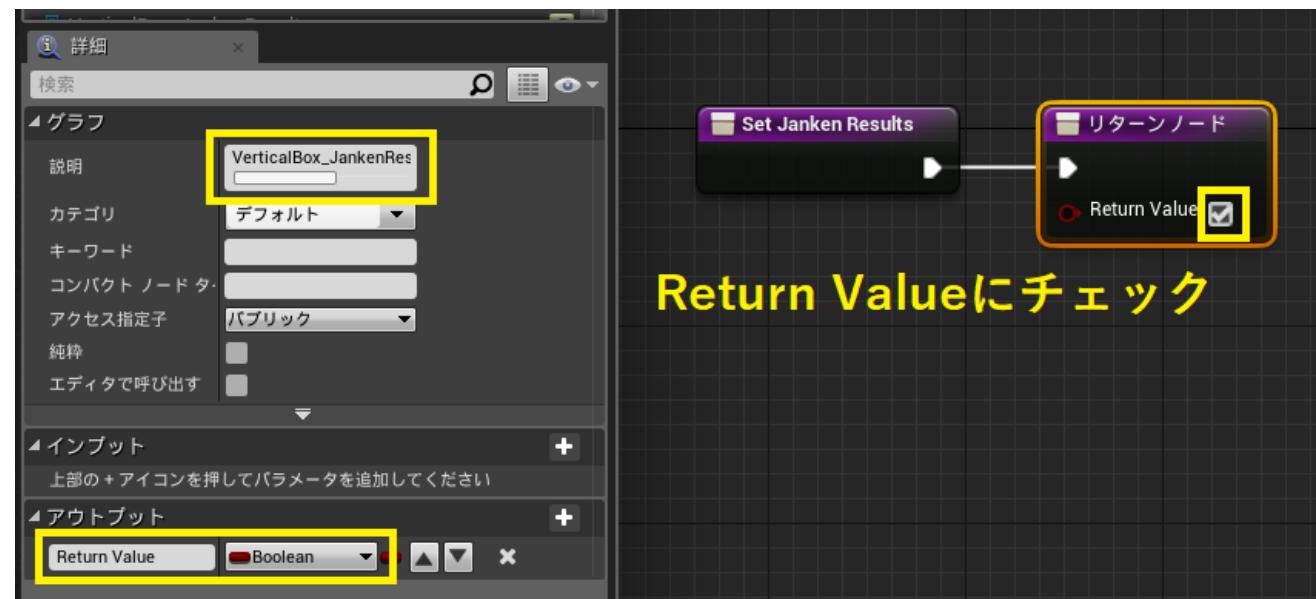
関数：SetJankenResultsを追加する 1



関数 : SetJankenResultsを追加する 2



関数：SetJankenResultsを追加する 3

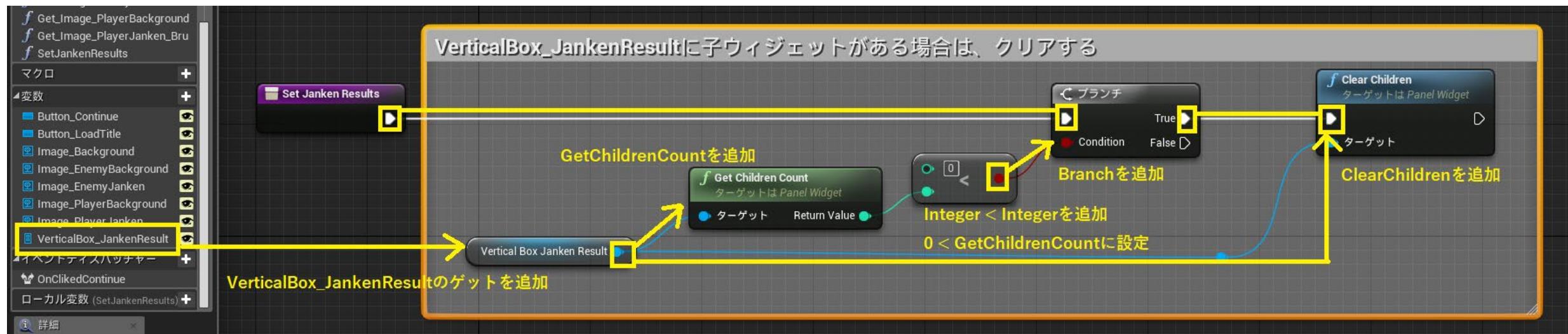


関数名	アクセス指定子	Input/Output	パラメータ名	型
SetJankenResults	パブリック	Output	ReturnValue	Boolean

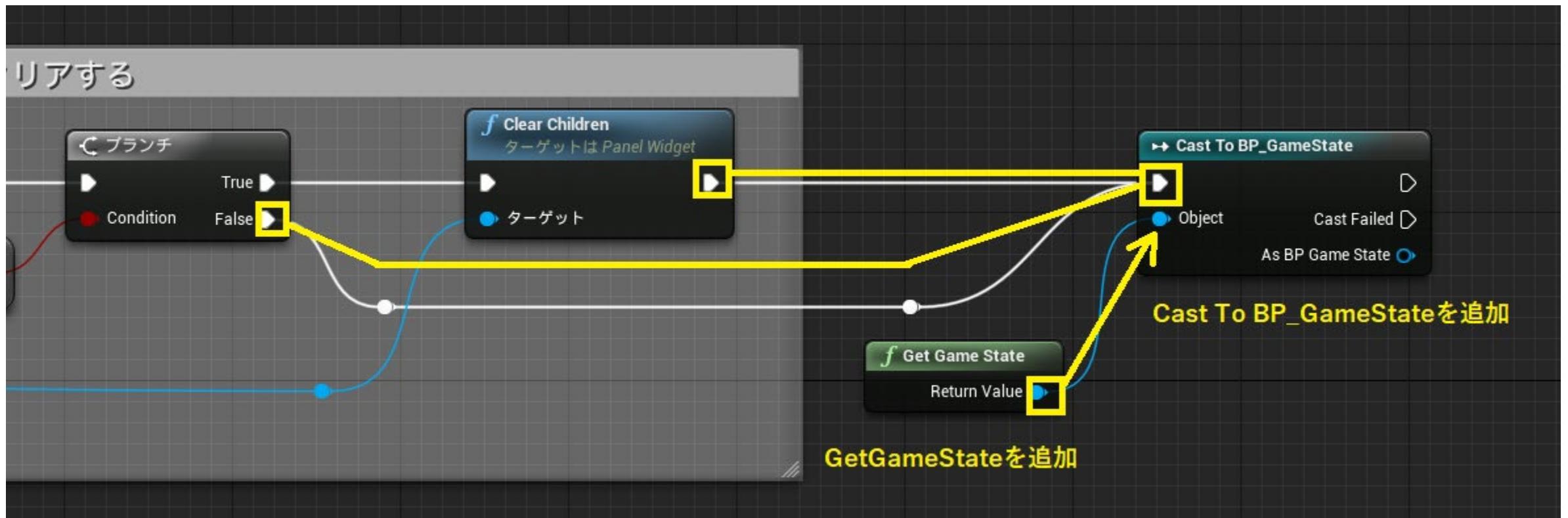
説明

VerticalBox_JankenResultにじゃんけんの結果を表示する

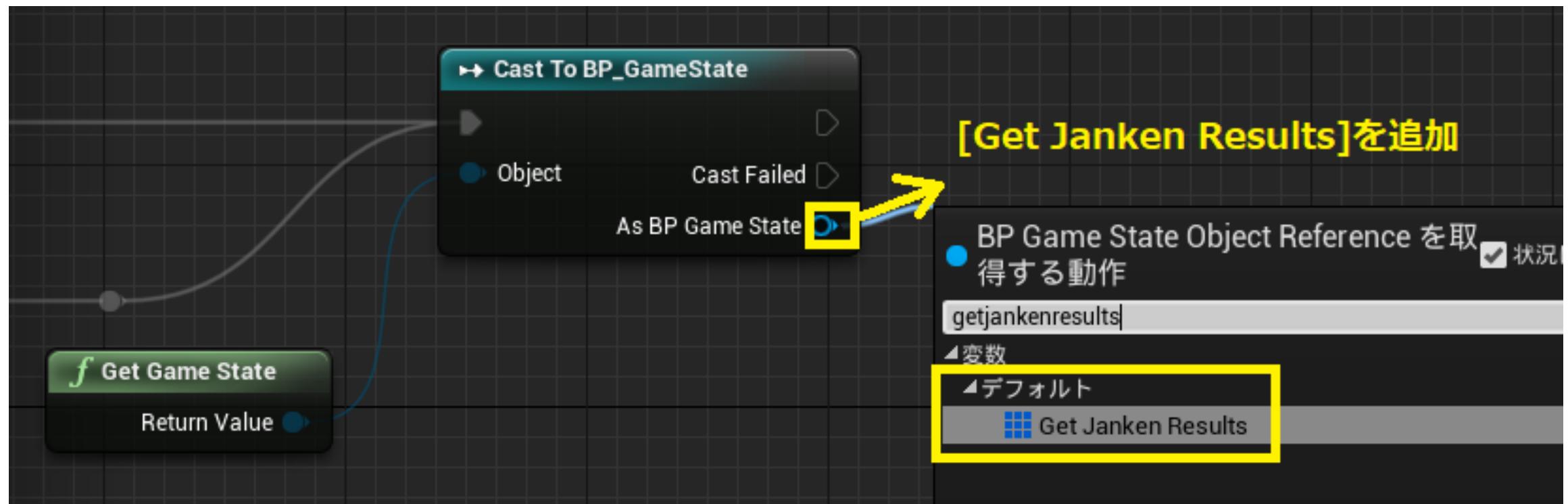
関数 : SetJankenResultsを追加する 4



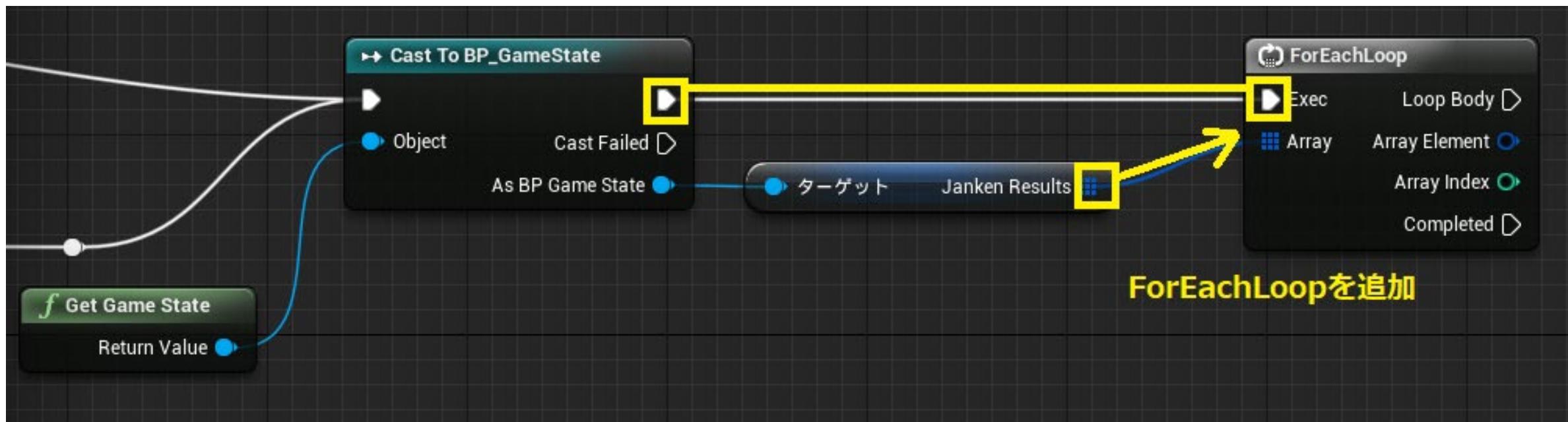
関数 : SetJankenResultsを追加する 5



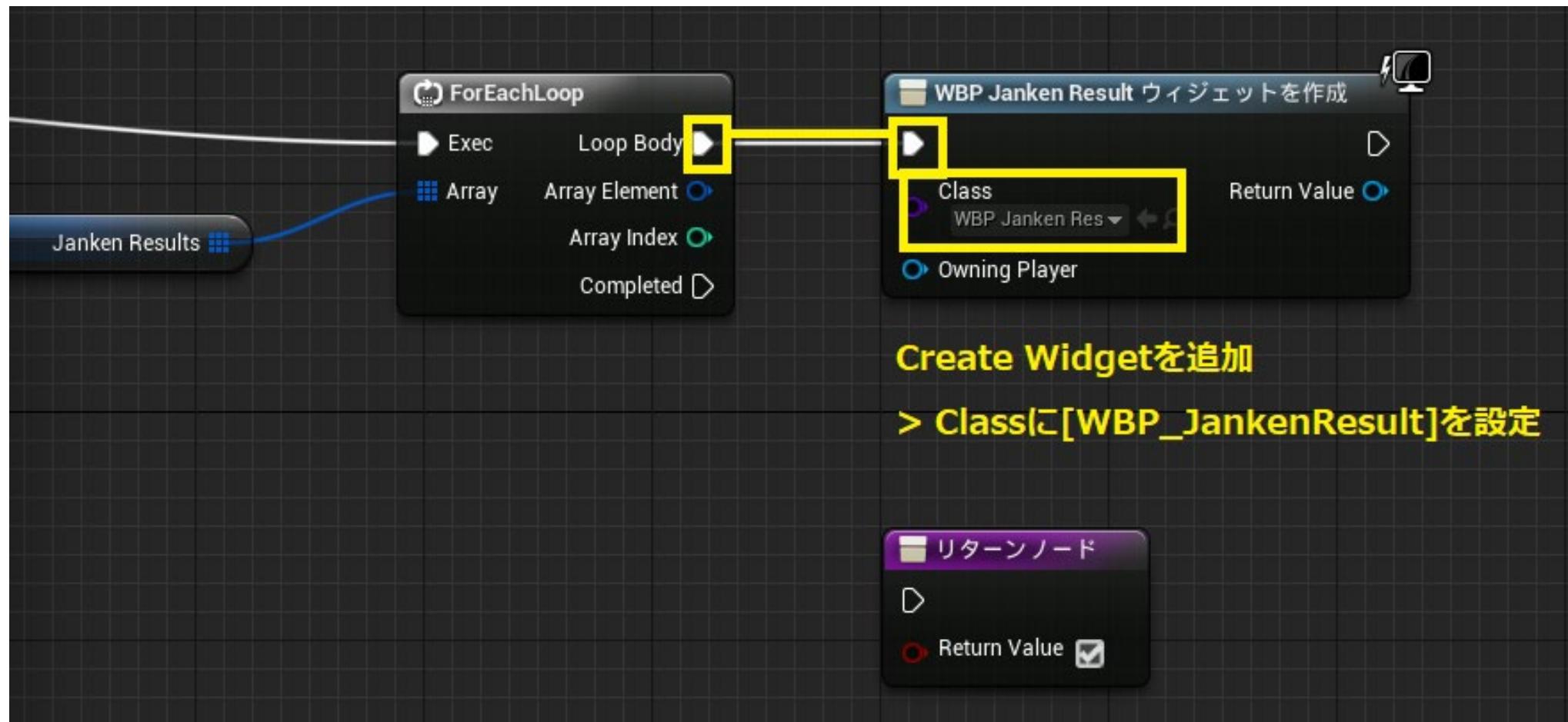
関数 : SetJankenResultsを追加する 6



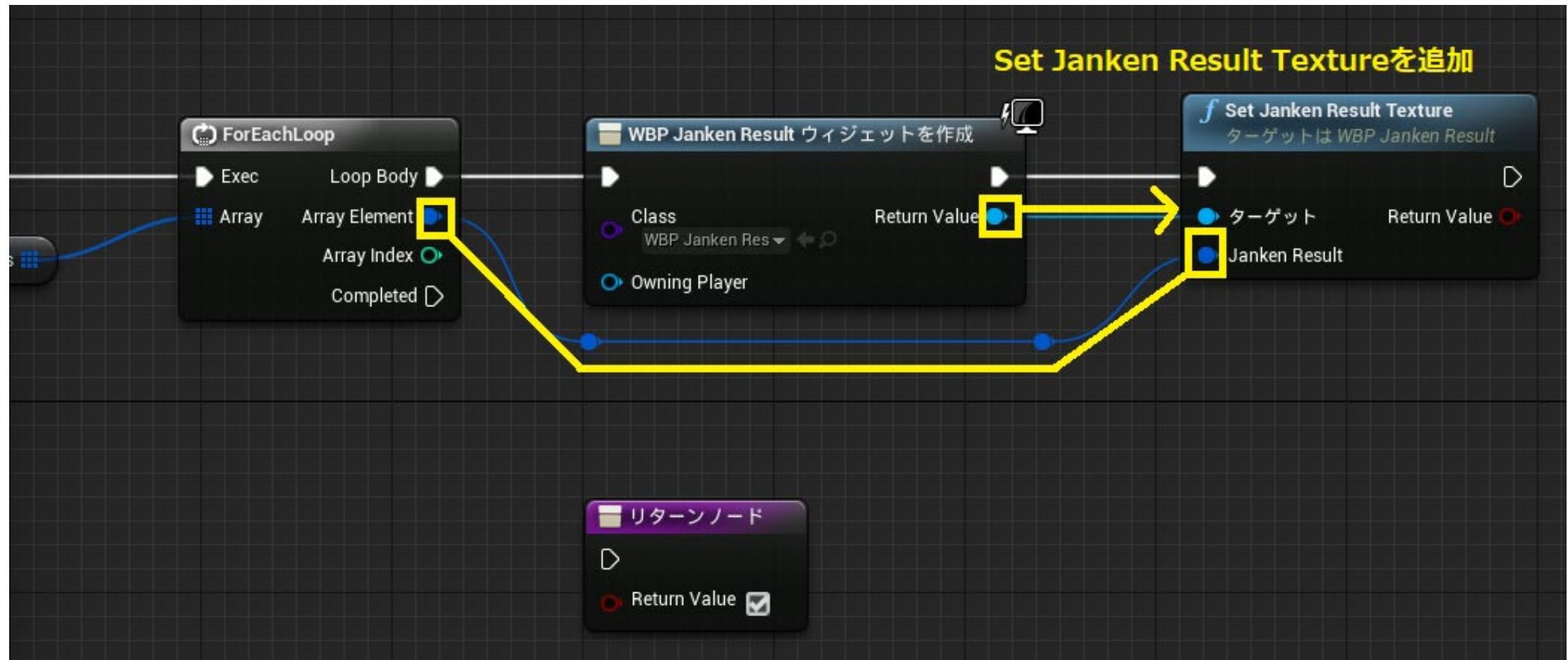
関数：SetJankenResultsを追加する 7



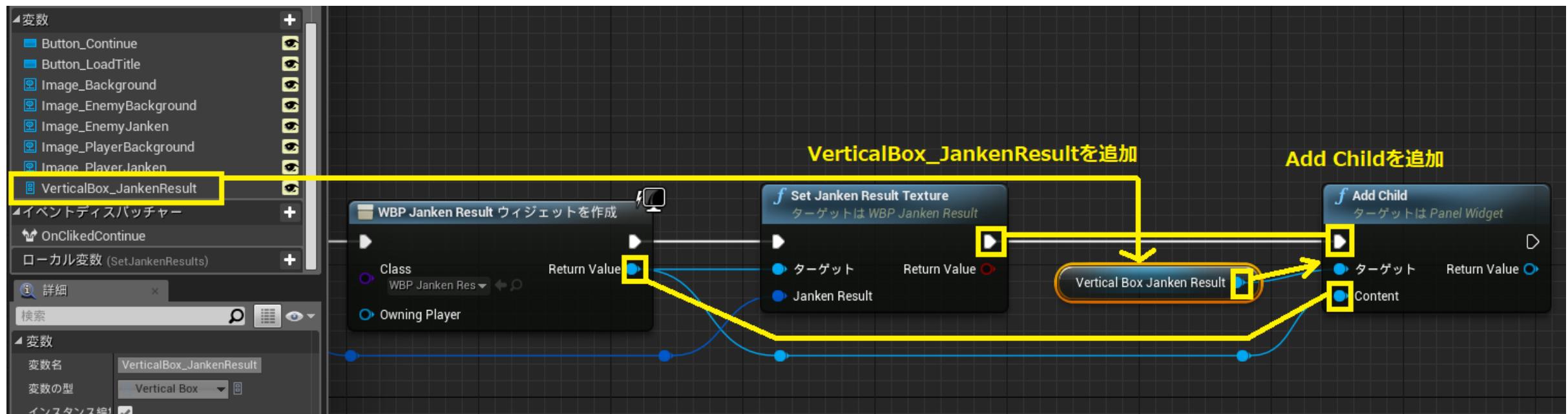
関数 : SetJankenResultsを追加する 8



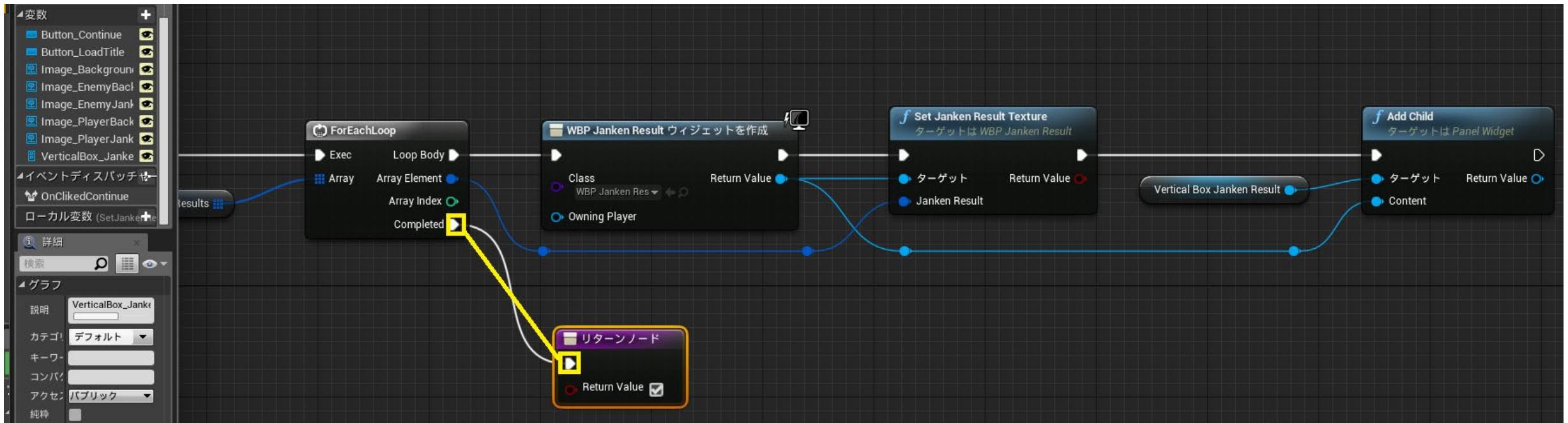
関数：SetJankenResultsを追加する 9



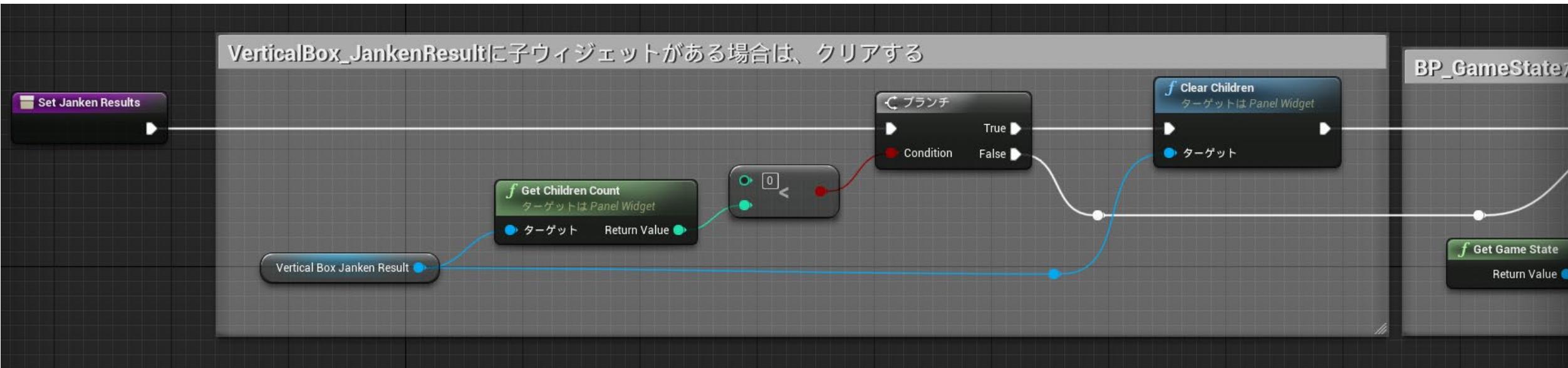
関数：SetJankenResultsを追加する 10



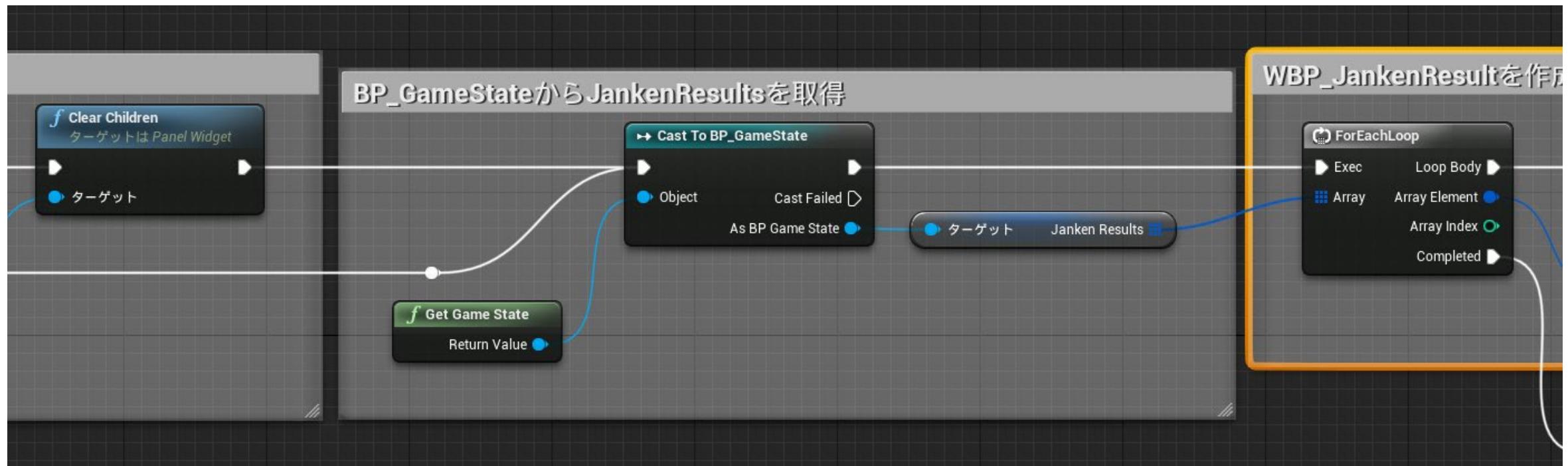
関数：SetJankenResultsを追加する 11



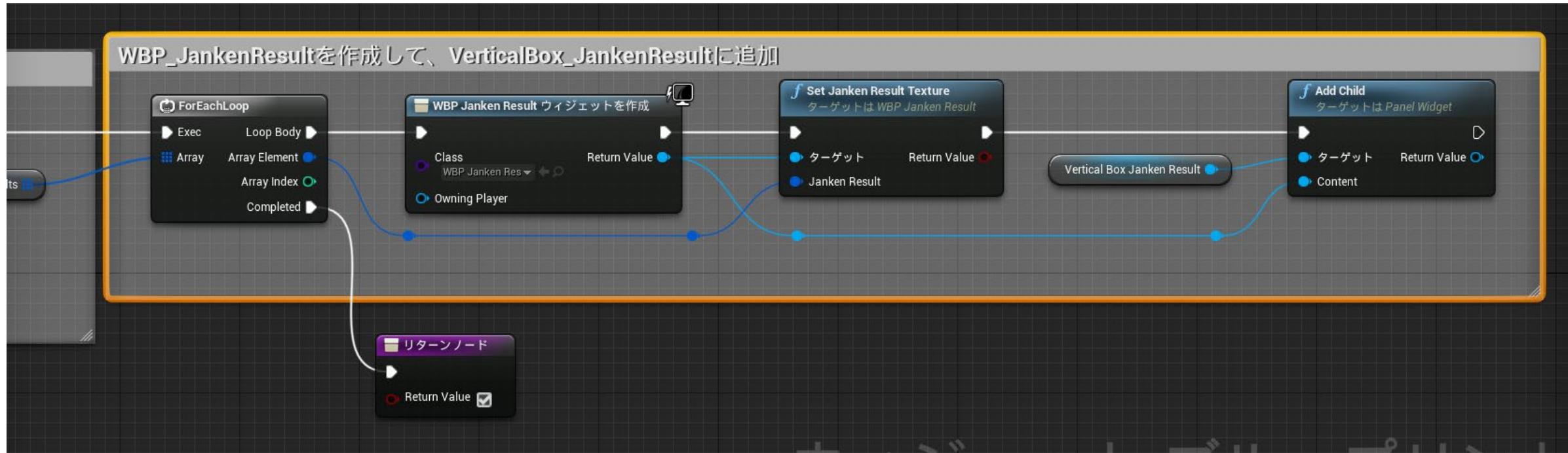
関数：SetJankenResultsを追加する 完成図 1



関数：SetJankenResultsを追加する 完成図 2



関数：SetJankenResultsを追加する 完成図 3



コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

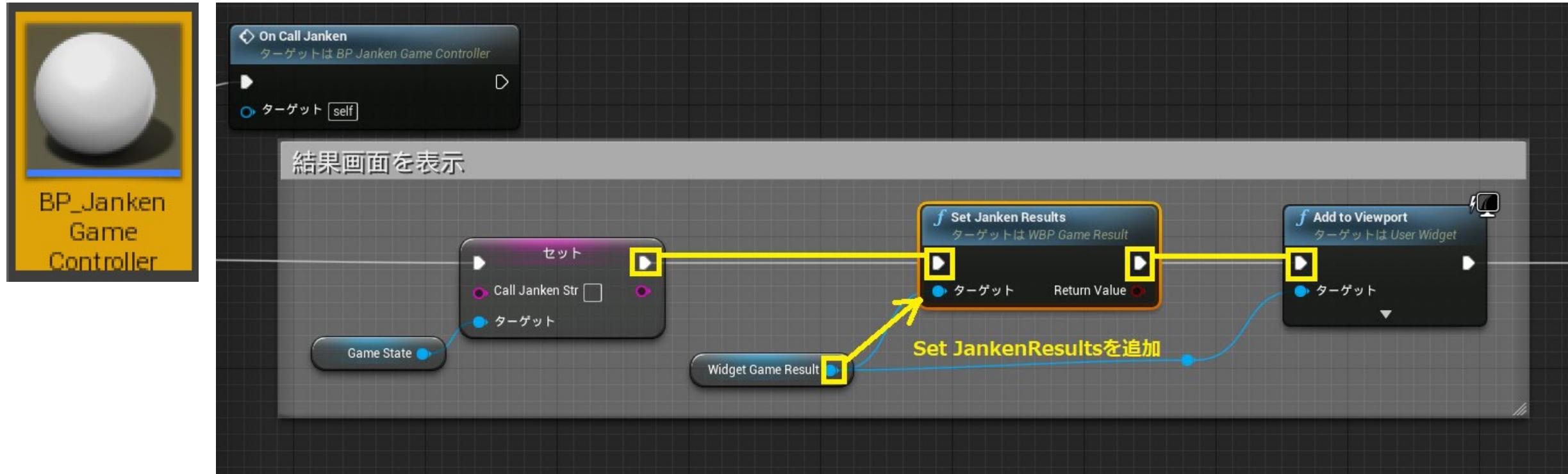
10.6.4 WBP_JankenResultに画像を設定する関数:SetJankenResultTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：SetJankenResultsを作成

10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

BP_JankenGameControllerからSetJankenResultsを呼び出して、
じゃんけんの結果を表示する



プレイして確認する



10. 結果を表示するUI作成

10.1 WBP_GameResultの作成

10.2 ウィジェットの配置

10.3 WBP_GameResultの表示

10.4 コンテニューボタンでWBP_GameResultを非表示にして、ゲームを再開

10.5 WBP_GameResultに結果情報を表示

10.6 FJankenResultを表示するWBP_JankenResultを作成

10.6.1 WBP_JankenResultのUIを作成

10.6.2 じゃんけんの画像を取得する関数:GetJankenTextureを作成

10.6.3 じゃんけんの勝敗画像を取得する関数:GetResultGameTextureを作成

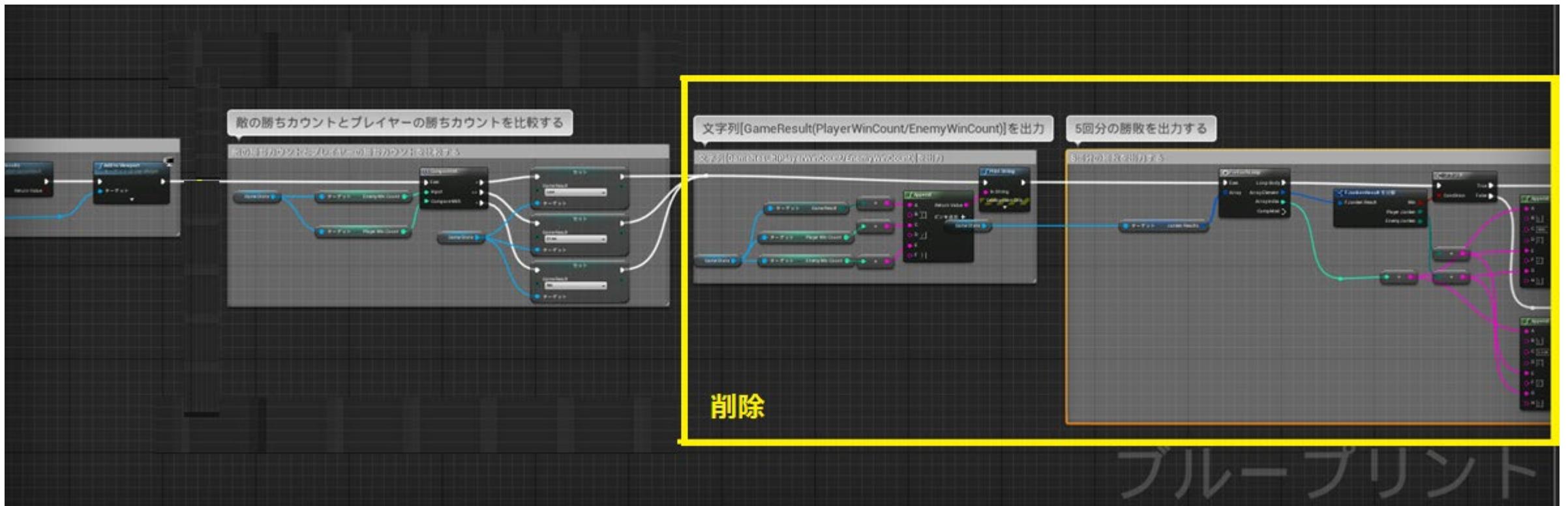
10.6.4 WBP_JankenResultに画像を設定する関数:SetJankenResultTextureを作成

10.7 WBP_GameResultのVerticalBox_JankenResultにWBP_JankenResultを表示する関数：SetJankenResultsを作成

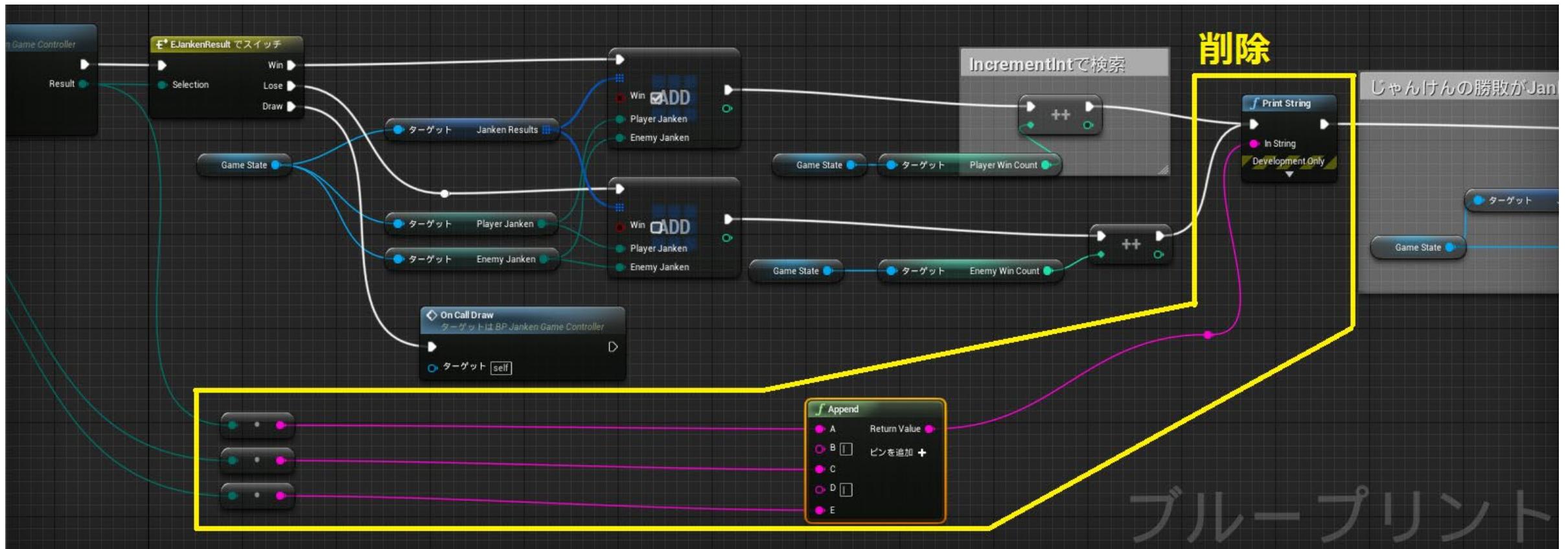
10.8 BP_JankenGameControllerからSetJankenResultsを呼び出して、じゃんけんの結果を表示する

10.9 PrintStringで出力している処理を削除

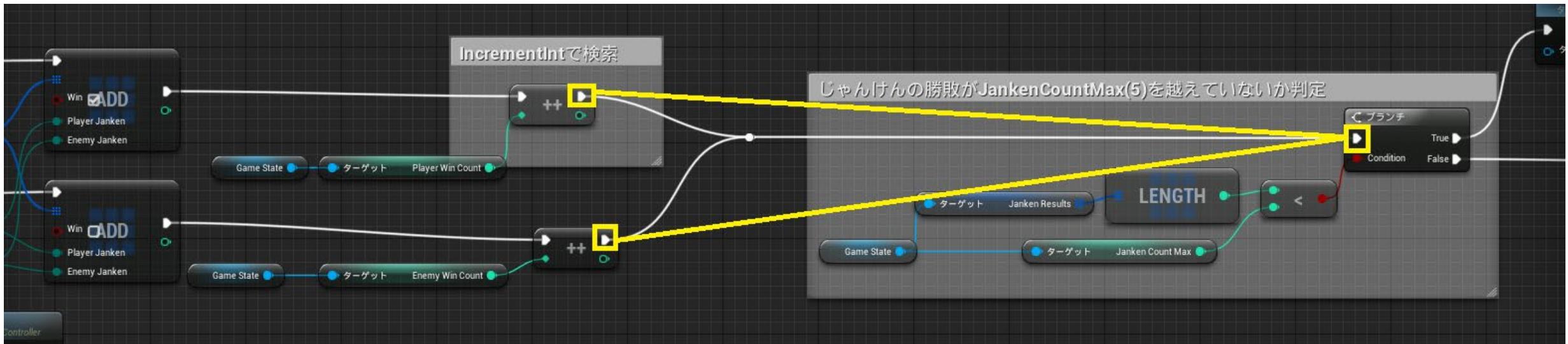
PrintStringで出力している処理を削除する 1



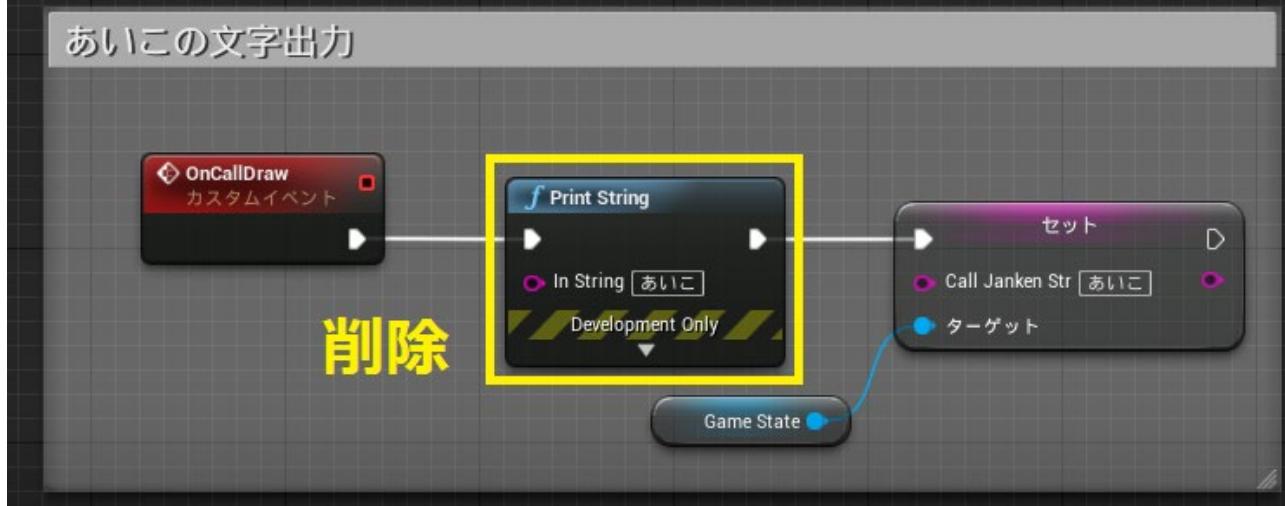
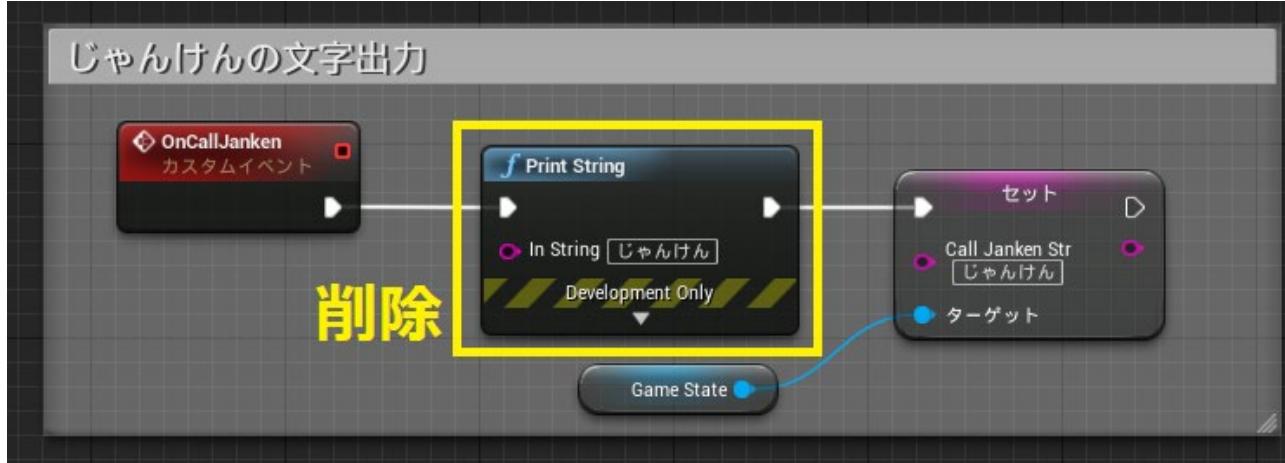
PrintStringで出力している処理を削除する 2



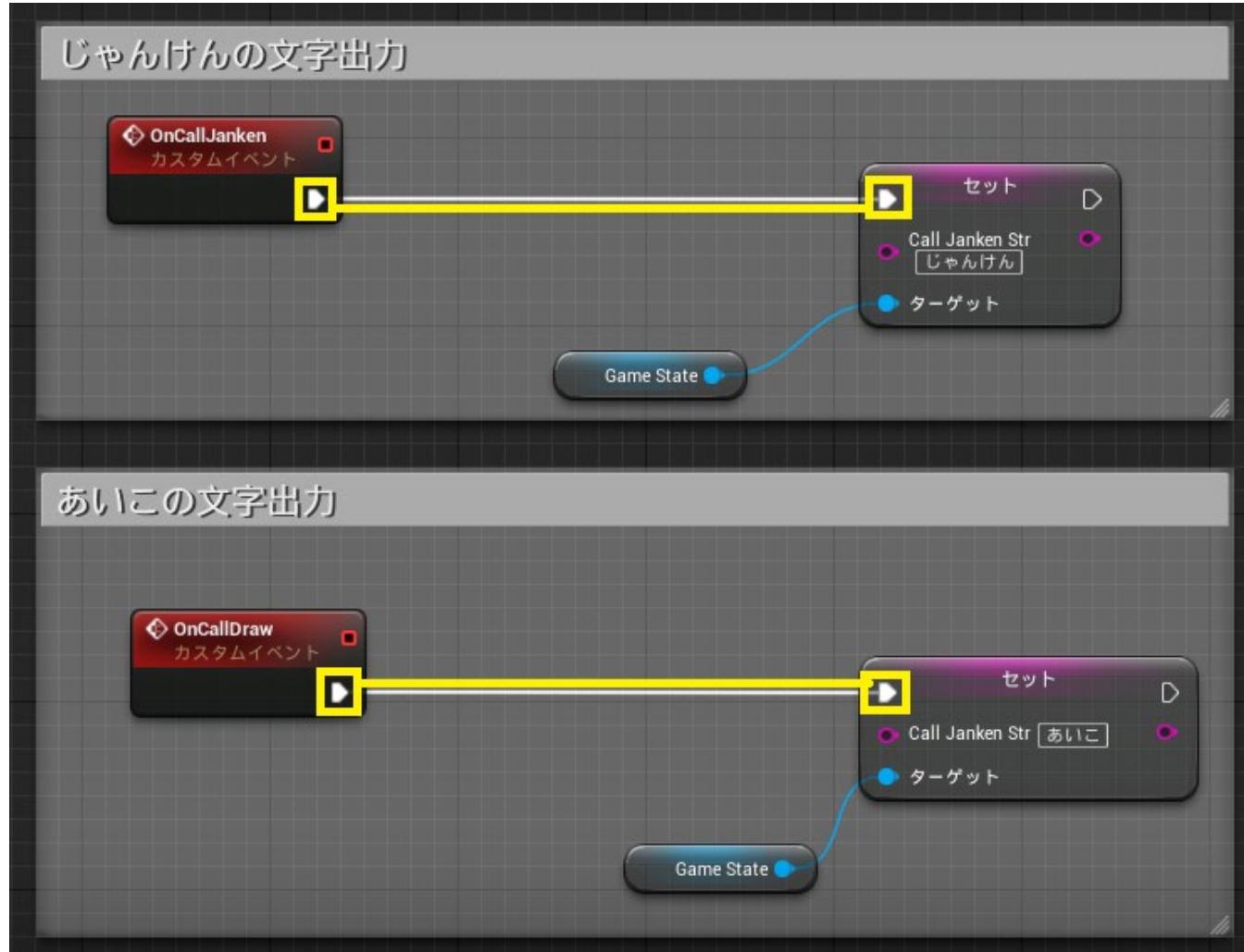
PrintStringで出力している処理を削除する 3



PrintStringで出力している処理を削除する 4



PrintStringで出力している処理を削除する 5



プレイして確認する



11. タイトルを表示するUI作成

11. タイトルを表示するUI作成

11.1 新規レベルTitleを作成

11.2 WBP_Titleの作成

11.3 ウィジェットの配置

11.4 ブループリント : BP_TitleControllerを作成

11.5 WBP_Titleにボタンのイベントディスパッチャを追加

11.6 BP_TitleControllerにボタンのイベントをバインド処理を実装

11.7 Loading画面 WBP>Loadingを作成

11.8 レベル遷移前にLoading画面を表示

11.9 WBP_GameResultのタイトルボタンからTitleレベルに遷移

11. タイトルを表示するUI作成

11.1 新規レベルTitleを作成

11.2 WBP_Titleの作成

11.3 ウィジェットの配置

11.4 ブループリント : BP_TitleControllerを作成

11.5 WBP_Titleにボタンのイベントディスパッチャを追加

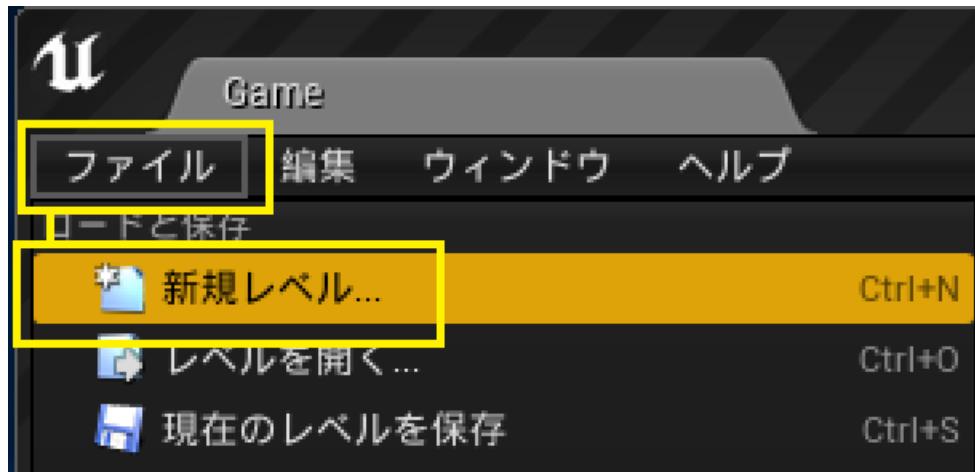
11.6 BP_TitleControllerにボタンのイベントをバインド処理を実装

11.7 Loading画面 WBP>Loadingを作成

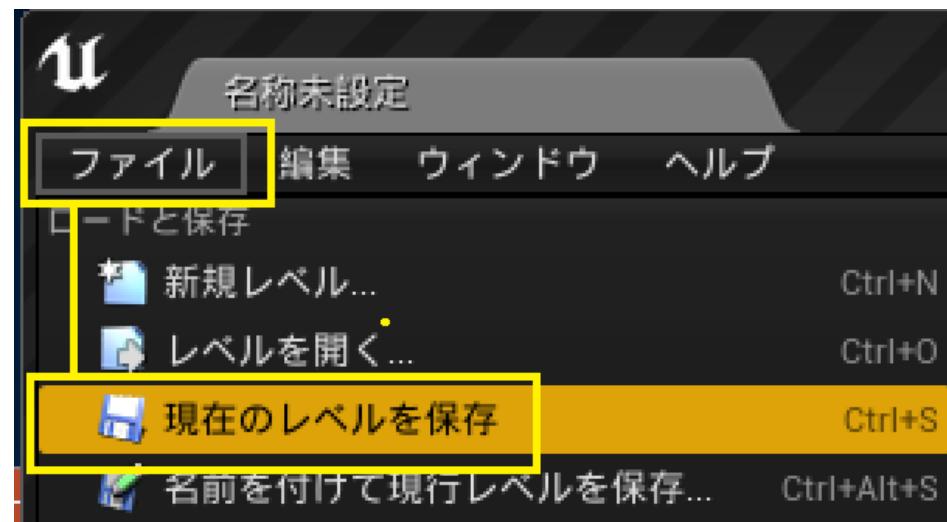
11.8 レベル遷移前にLoading画面を表示

11.9 WBP_GameResultのタイトルボタンからTitleレベルに遷移

新規レベル:Titleを作成する 1



新規レベル:Titleを作成する 2



11. タイトルを表示するUI作成

11.1 新規レベルTitleを作成

11.2 WBP_Titleの作成

11.3 ウィジェットの配置

11.4 ブループリント : BP_TitleControllerを作成

11.5 WBP_Titleにボタンのイベントディスパッチャを追加

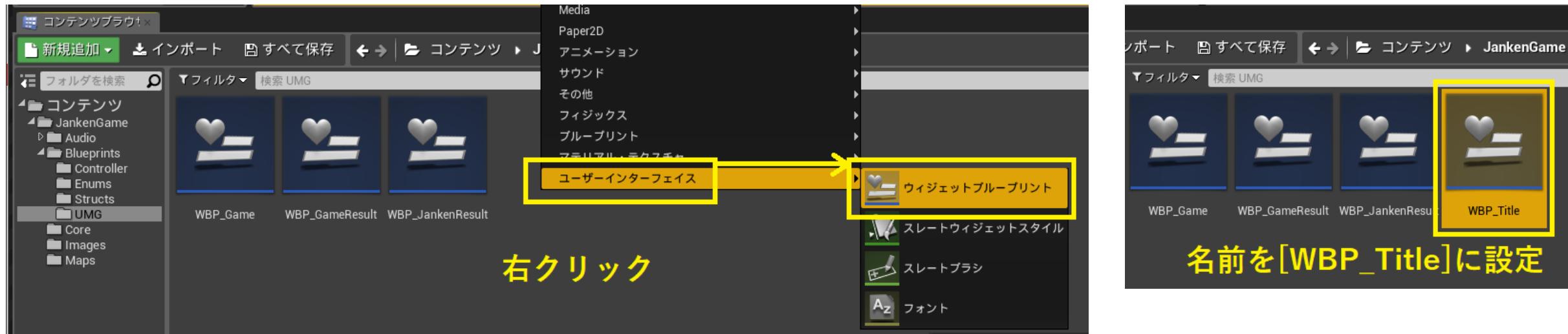
11.6 BP_TitleControllerにボタンのイベントをバインド処理を実装

11.7 Loading画面 WBP>Loadingを作成

11.8 レベル遷移前にLoading画面を表示

11.9 WBP_GameResultのタイトルボタンからTitleレベルに遷移

ウィジェットブループリント:WBP_Titleを作成する



11. タイトルを表示するUI作成

11.1 新規レベルTitleを作成

11.2 WBP_Titleの作成

11.3 ウィジェットの配置

11.4 ブループリント : BP_TitleControllerを作成

11.5 WBP_Titleにボタンのイベントディスパッチャを追加

11.6 BP_TitleControllerにボタンのイベントをバインド処理を実装

11.7 Loading画面 WBP>Loadingを作成

11.8 レベル遷移前にLoading画面を表示

11.9 WBP_GameResultのタイトルボタンからTitleレベルに遷移

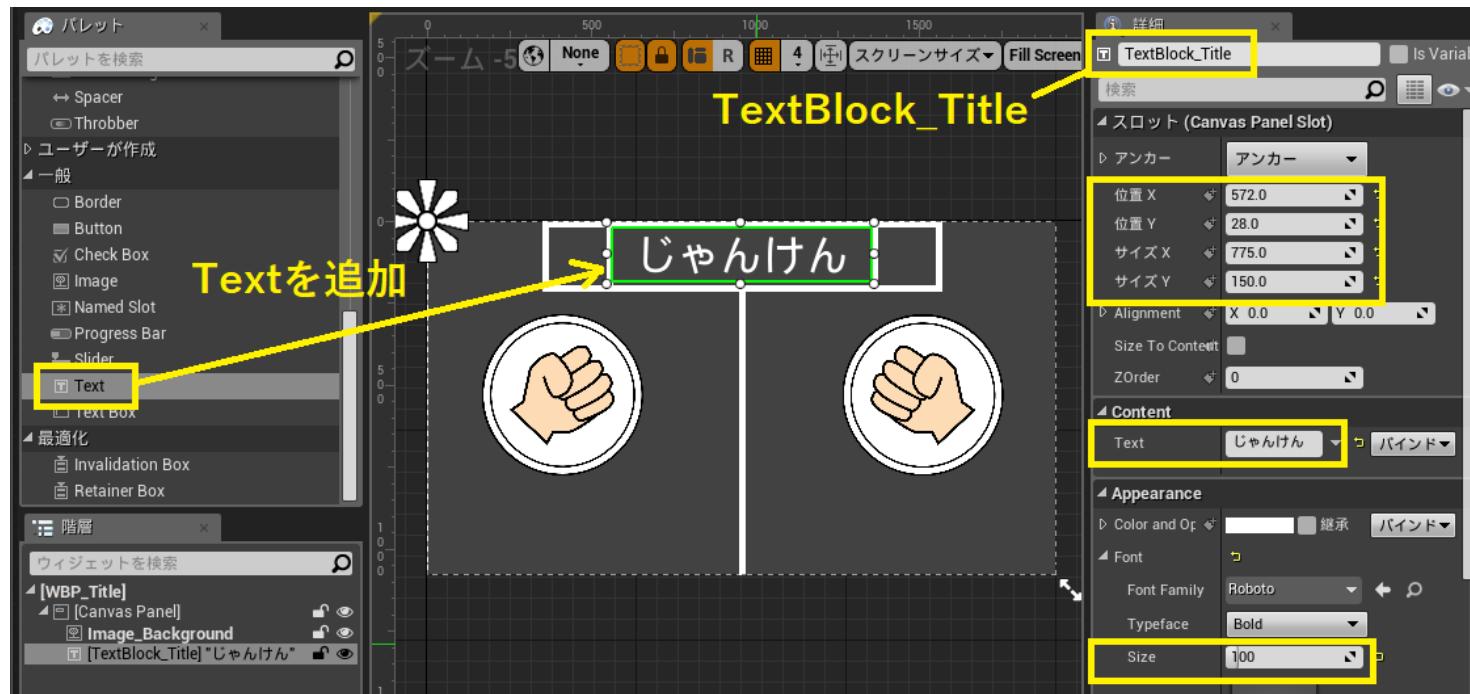
Image:Image_Backgroundを追加する

The screenshot shows the Construct 3 editor interface with the following components:

- Palette (Left):** Shows various UI elements like Vertical Box, Wrap Box, and Image.
- Canvas (Center):** Displays a vertical stack of three "Image" components. The top one is highlighted with a green dashed border and labeled "Image_Background". The middle and bottom ones are labeled "T_Background_Title".
- Properties Panel (Right):** Shows the properties for "Image_Background".
 - Image_Background:** Variable name.
 - Anchor:** Set to "Anchor".
 - Offset:** All four offsets (Left, Top, Right, Bottom) are set to 0.0.
 - Appearance:** Set to "Image".
 - Image Size:** X: 1920.0, Y: 1080.0.
 - Draw As:** Set to "Image".
 - Tiling:** Set to "No Tile".
 - Color and Opacity:** Set to 100%.
- Inspector Panel (Bottom Right):** Shows the component tree:
 - Image_Background (selected)
 - Image
 - Image
 - ImageA tooltip indicates: "Hold [Shift] to update the alignment to match. Hold [Control] to update the position to match."
- Properties Table (Bottom Right):** A summary of the component properties.

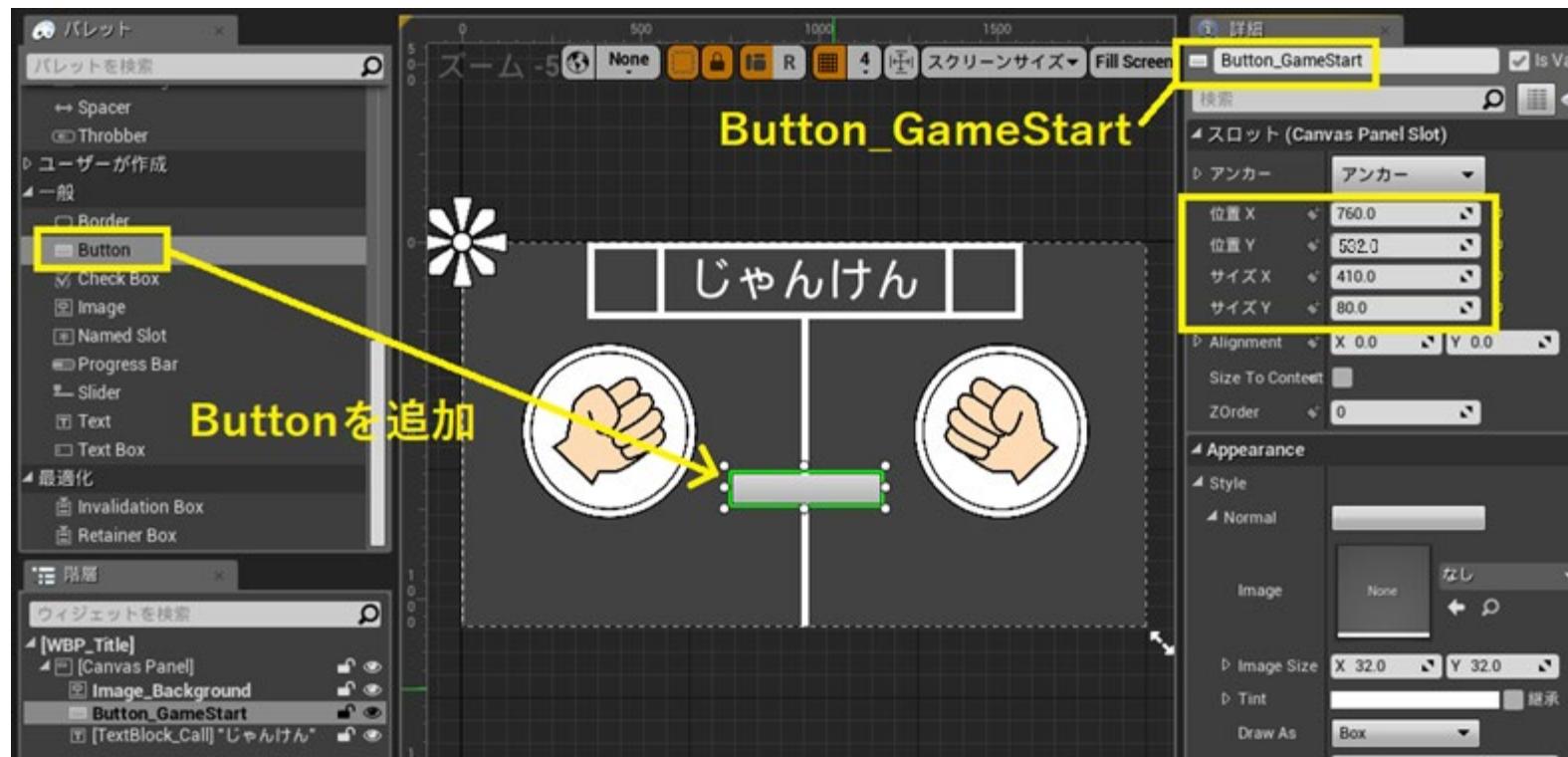
プロパティ	値
変数名	Image_Background
左オフセット	0.0
上オフセット	0.0
右オフセット	0.0
下オフセット	0.0
Brush > Image	T_Background_Title

Text:TextBlock_Titleを追加する



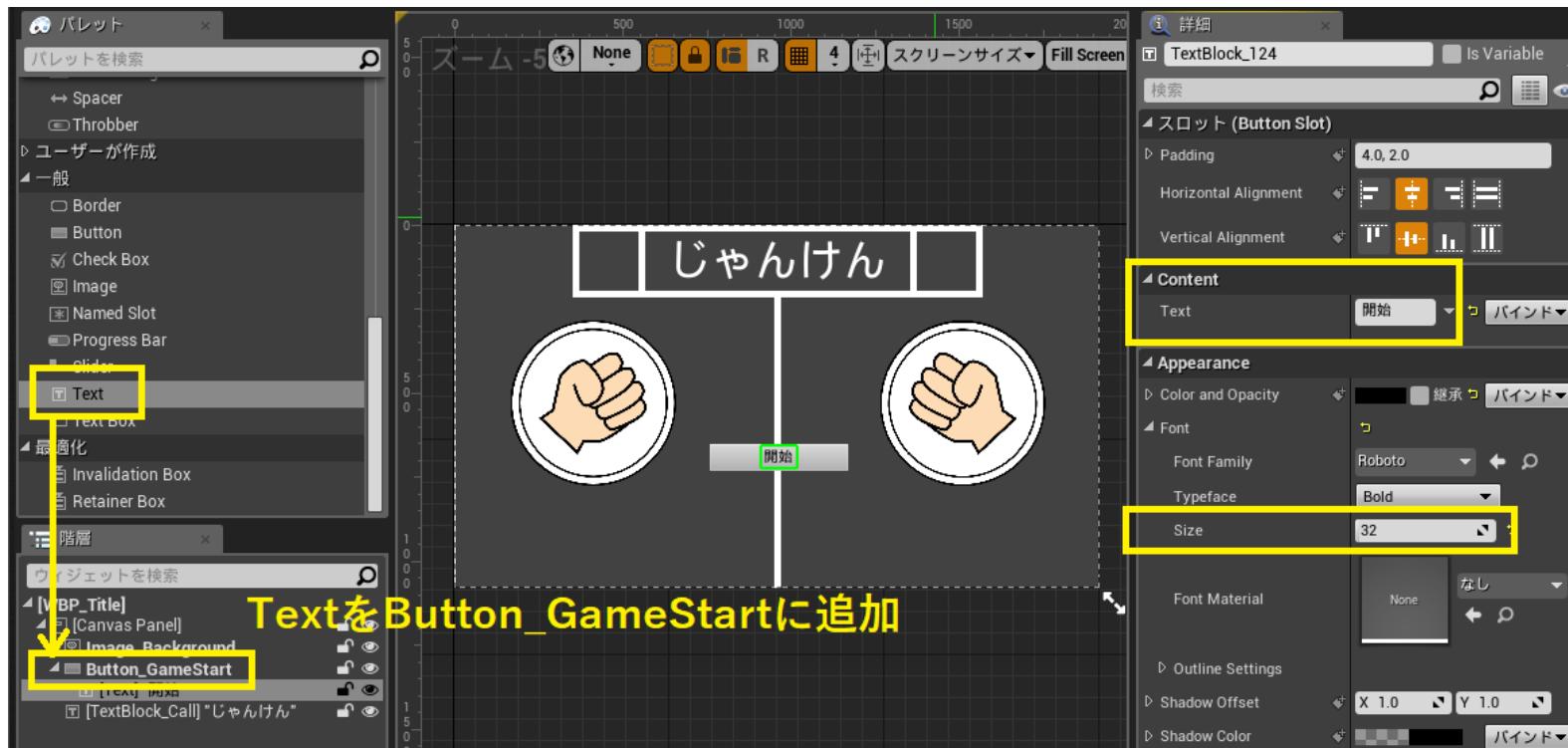
プロパティ	値
変数名	TextBlock_Title
位置X	572.0
位置Y	28.0
サイズX	775.0
サイズY	150.0
Text	じゃんけん
Font > Size	100
Justification	テキスト中央揃え

Button:Button_GameStartを追加する



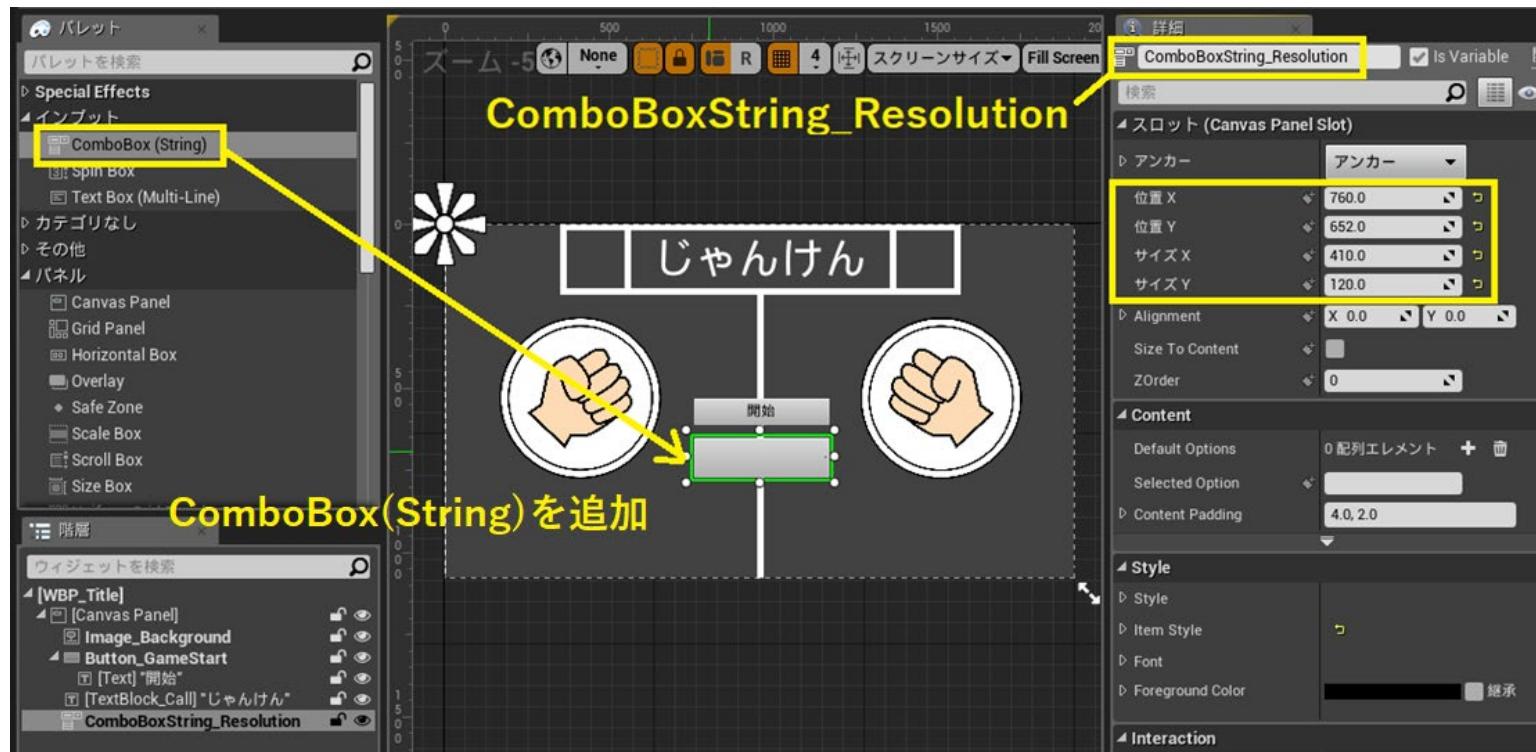
プロパティ	値
変数名	Button_GameStart
位置X	760.0
位置Y	532.0
サイズX	410.0
サイズY	80.0

TextをButton_GameStartに追加する



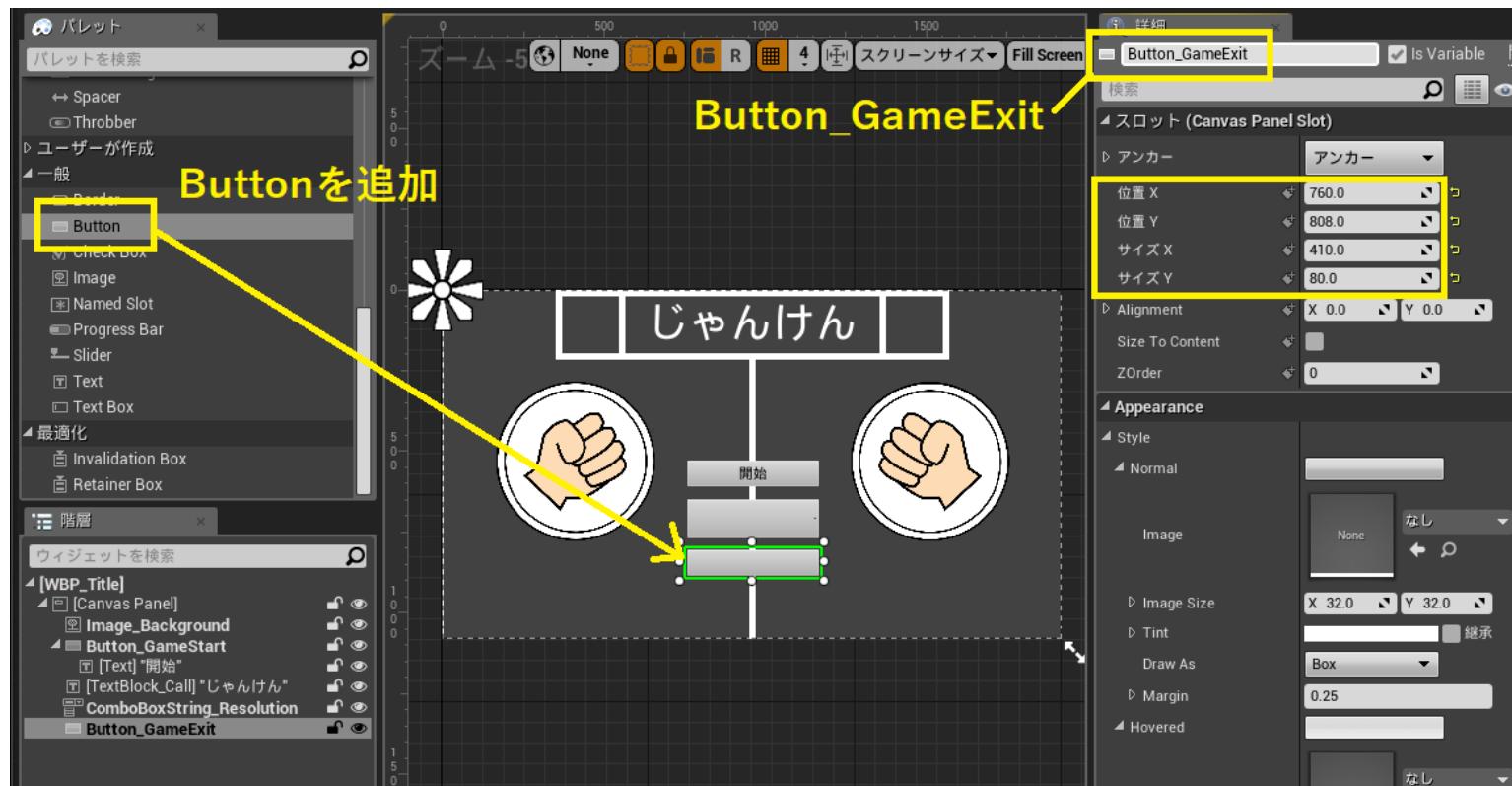
プロパティ	値
Text	開始
Color and Opacity	黒(R:0.0 G:0.0 B:0.0 A:1.0)
Font > Size	32

ComboBox(String):ComboBoxString_Resolutionを追加する



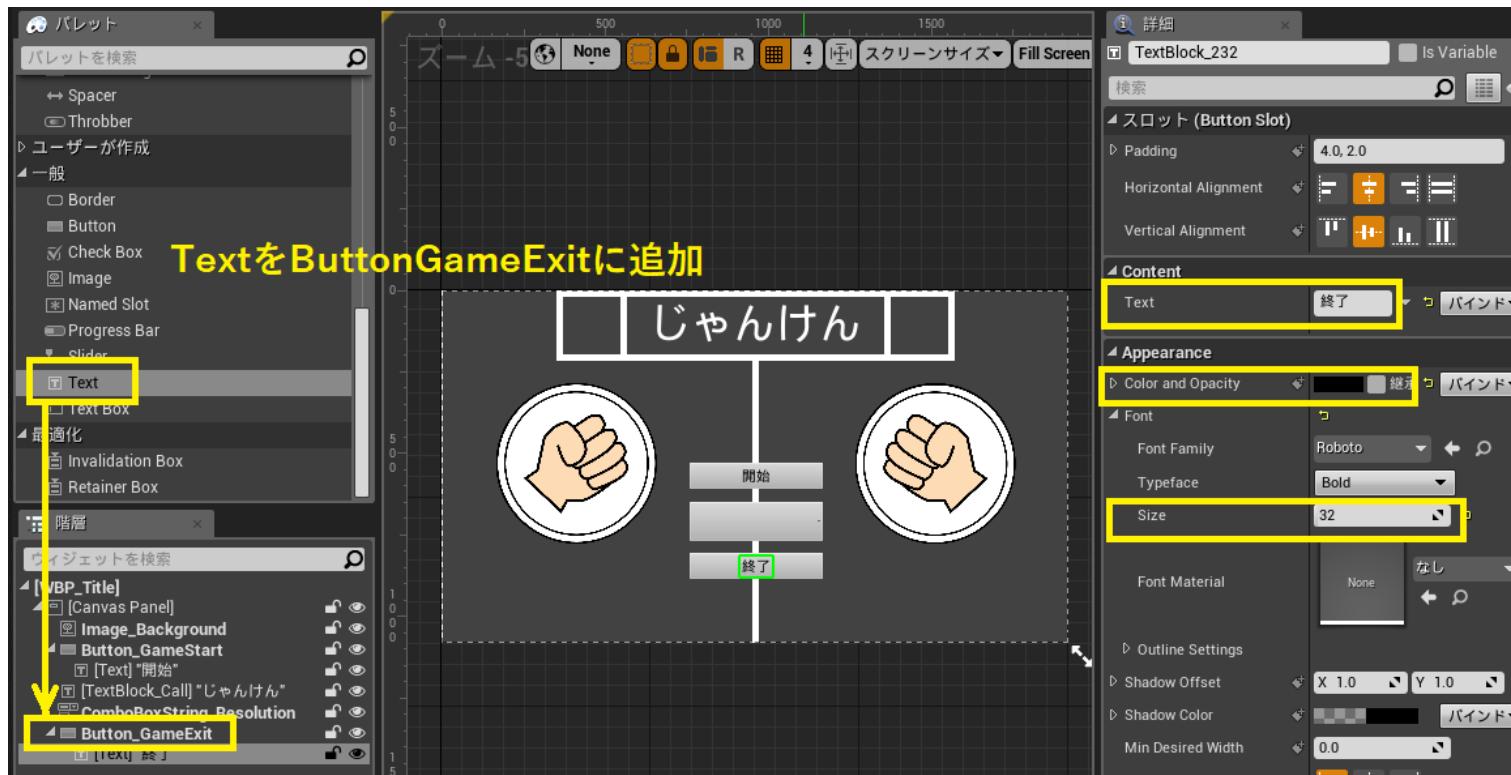
プロパティ	値
変数名	ComboBoxString_Resolution
位置X	760.0
位置Y	652.0
サイズX	410.0
サイズY	120.0

Button:Button_GameExitを追加する



プロパティ	値
変数名	Button_GameExit
位置X	760.0
位置Y	808.0
サイズX	410
サイズY	80

TextをButtonGameExitに追加する



プロパティ	値
Text	終了
Color and Opacity	黒(R:0.0 G:0.0 B:0.0 A:1.0)
Font > Size	32

11. タイトルを表示するUI作成

11.1 新規レベルTitleを作成

11.2 WBP_Titleの作成

11.3 ウィジェットの配置

11.4 ブループリント : BP_TitleControllerを作成

11.5 WBP_Titleにボタンのイベントディスパッチャを追加

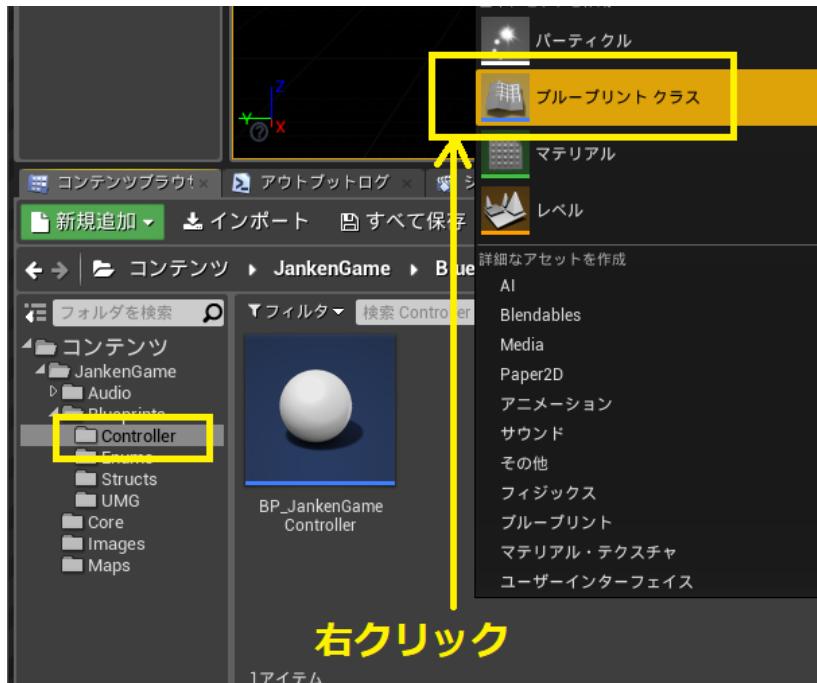
11.6 BP_TitleControllerにボタンのイベントをバインド処理を実装

11.7 Loading画面 WBP>Loadingを作成

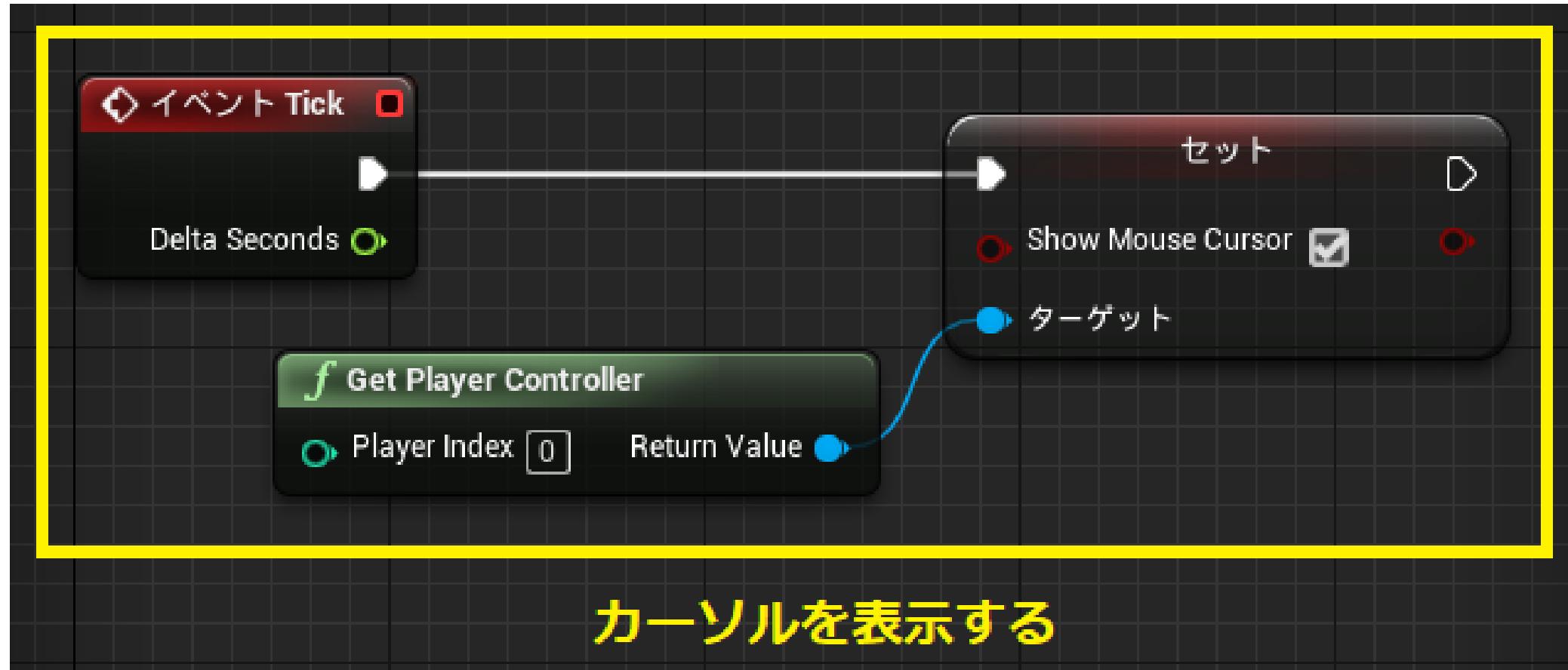
11.8 レベル遷移前にLoading画面を表示

11.9 WBP_GameResultのタイトルボタンからTitleレベルに遷移

ブループリント:BP_TitleController(親:Actor)を作成する

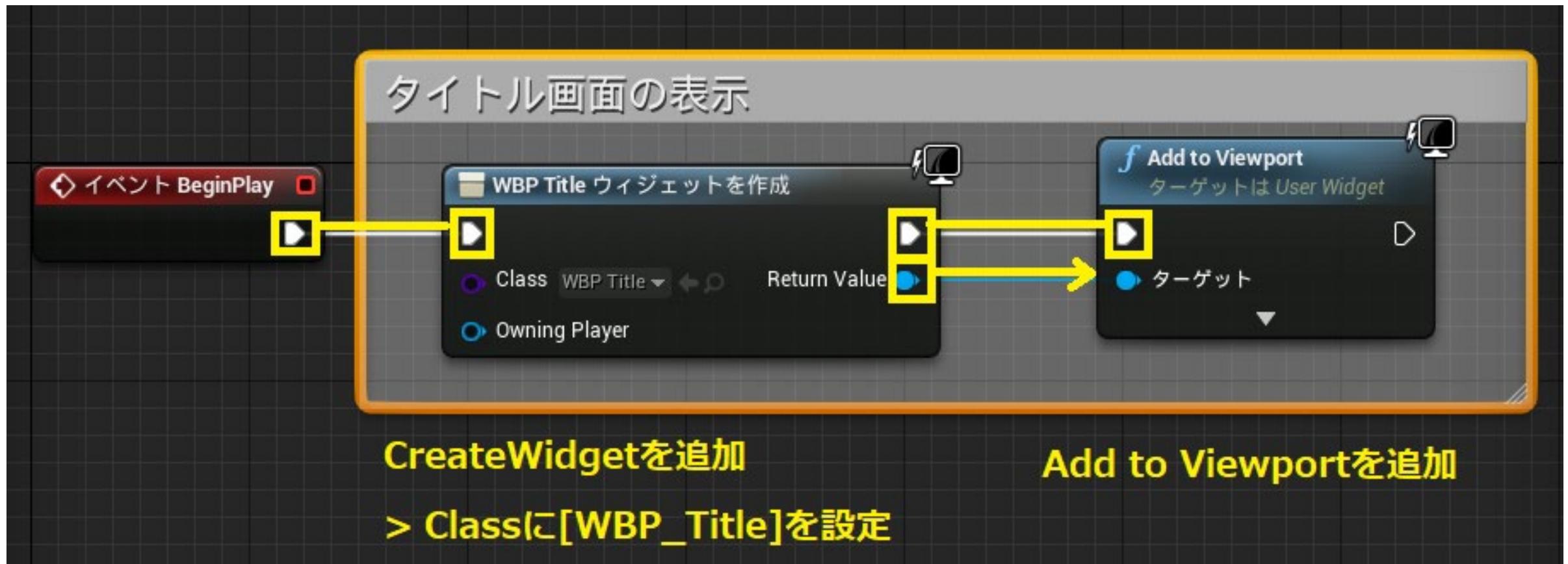


Tickでマウスカーソルを常に表示する処理を実装する

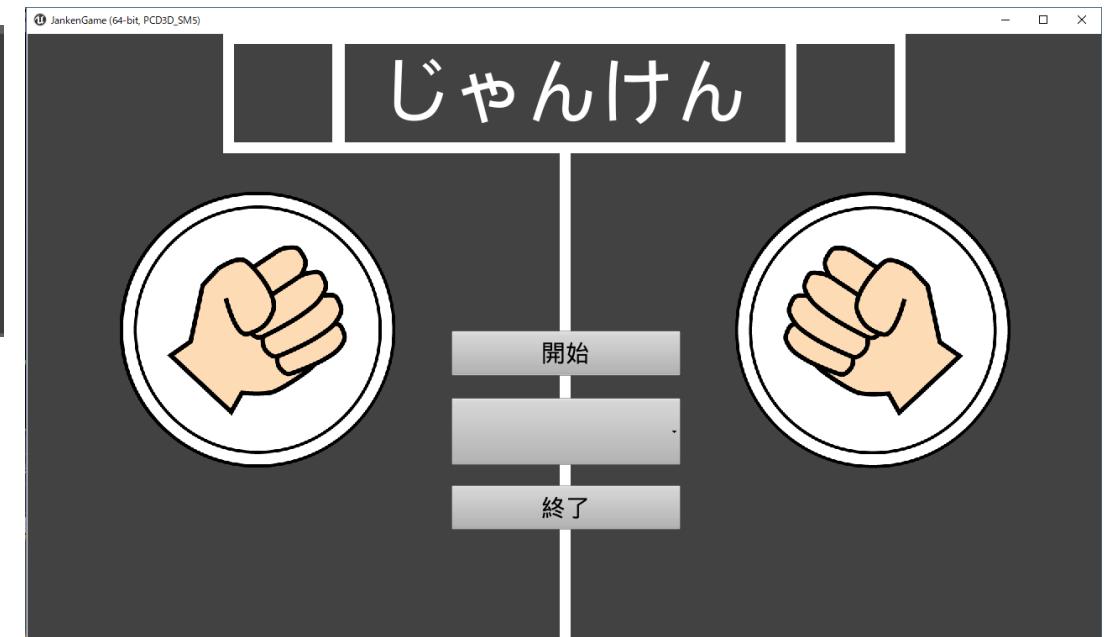
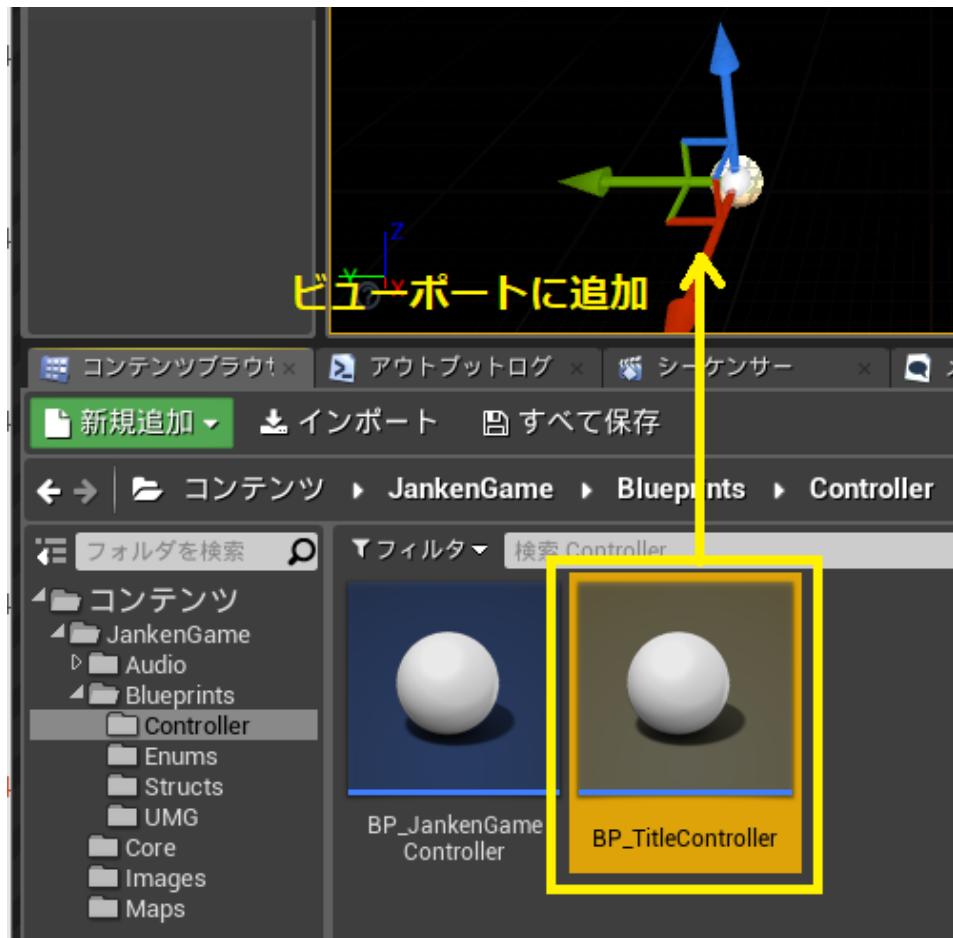


カーソルを表示する

BeginPlayでWBP_Titleを表示する



プレイして確認する



11. タイトルを表示するUI作成

11.1 新規レベルTitleを作成

11.2 WBP_Titleの作成

11.3 ウィジェットの配置

11.4 ブループリント : BP_TitleControllerを作成

11.5 WBP_Titleにボタンのイベントディスパッチャを追加

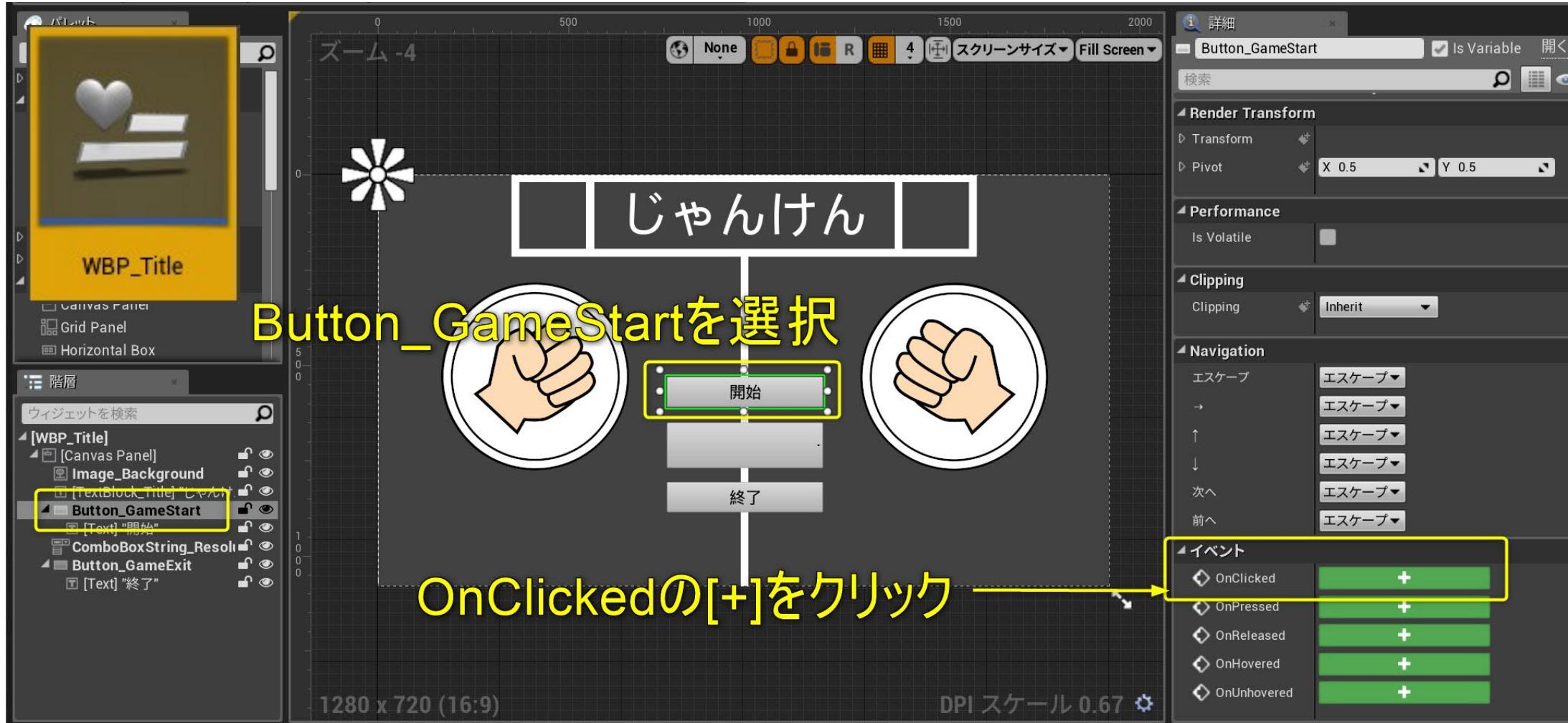
11.6 BP_TitleControllerにボタンのイベントをバインド処理を実装

11.7 Loading画面 WBP>Loadingを作成

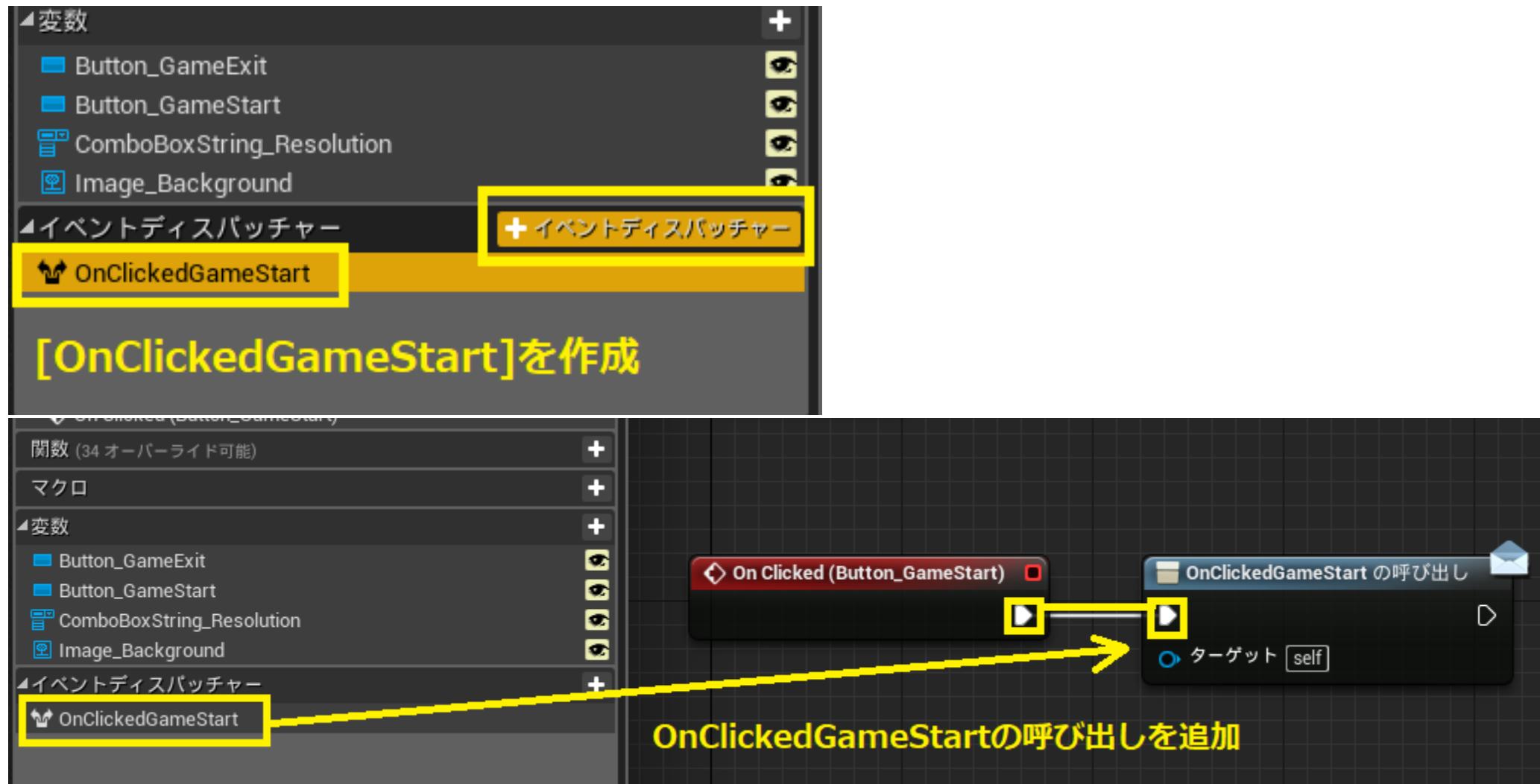
11.8 レベル遷移前にLoading画面を表示

11.9 WBP_GameResultのタイトルボタンからTitleレベルに遷移

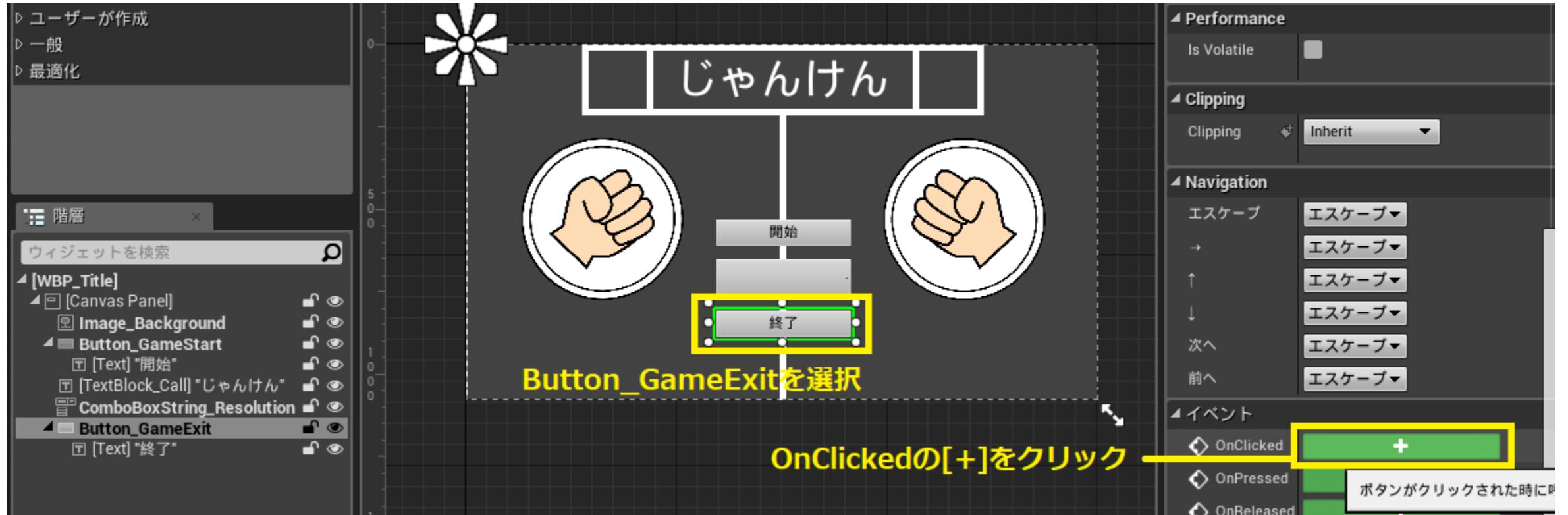
WBP_Title Button_GameStartにOnClickedイベントを追加する 1



WBP_Title Button_GameStartにOnClickedイベントを追加する 2
イベントディスパッチャー:OnClickedGameStartを追加する



WBP_Title Button_GameExitにOnClickedイベントを追加する 1



WBP_Title Button_GameExitにOnClickedイベントを追加する
2
イベントディスパッチャー：OnClickedGameExitを追加する



11. タイトルを表示するUI作成

11.1 新規レベルTitleを作成

11.2 WBP_Titleの作成

11.3 ウィジェットの配置

11.4 ブループリント : BP_TitleControllerを作成

11.5 WBP_Titleにボタンのイベントディスパッチャを追加

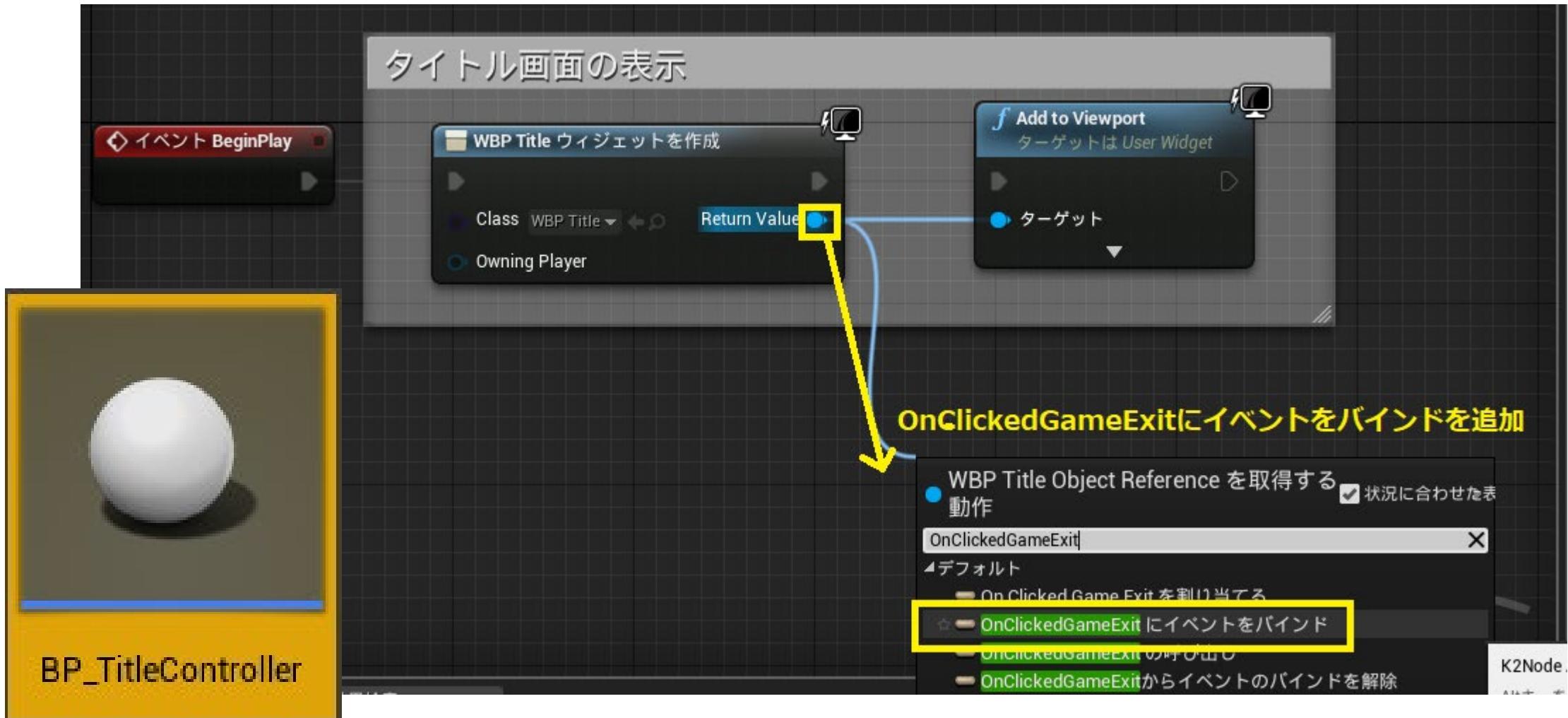
11.6 BP_TitleControllerにボタンのイベントをバインド処理を実装

11.7 Loading画面 WBP>Loadingを作成

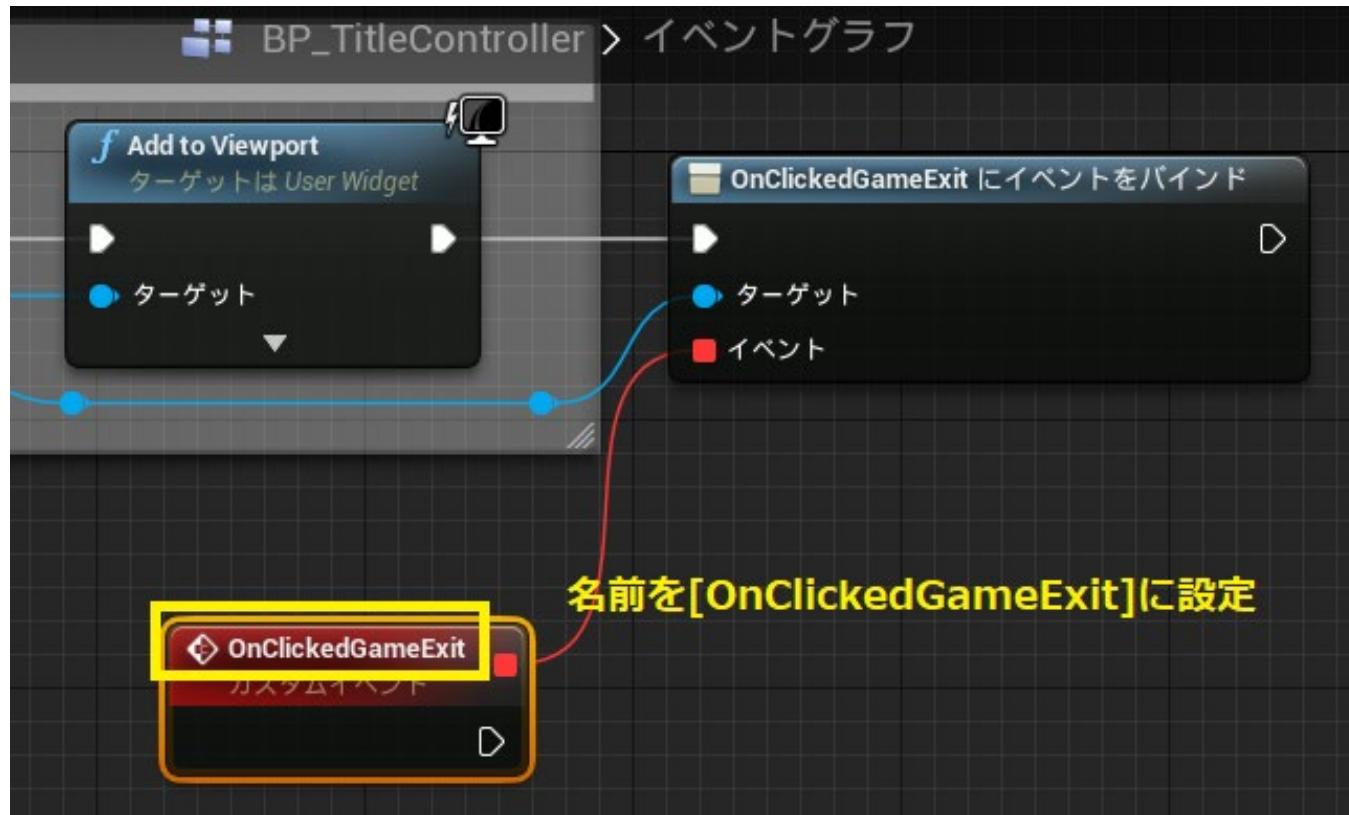
11.8 レベル遷移前にLoading画面を表示

11.9 WBP_GameResultのタイトルボタンからTitleレベルに遷移

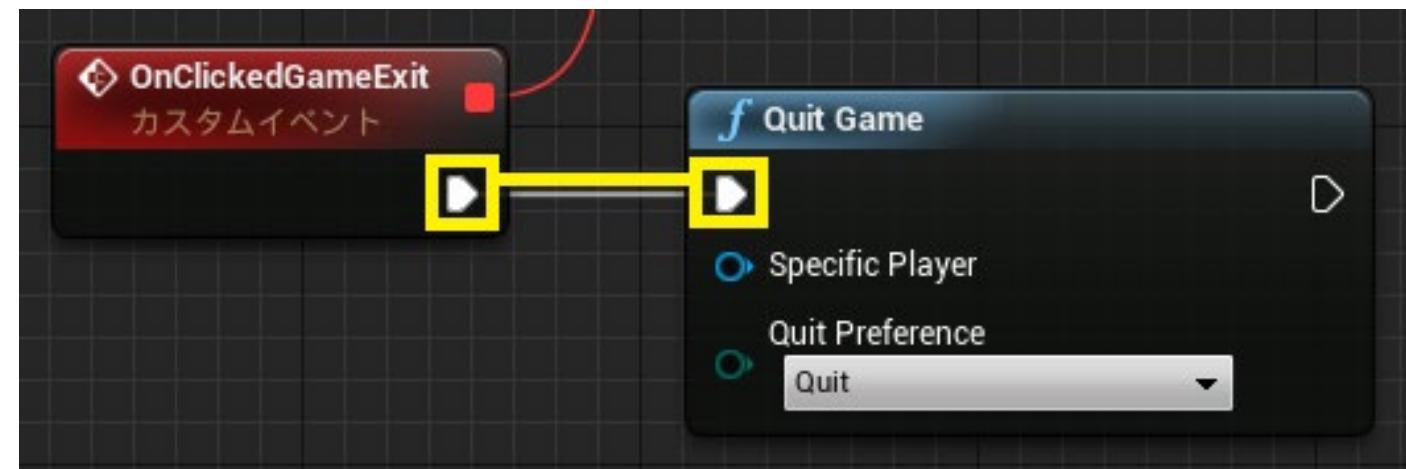
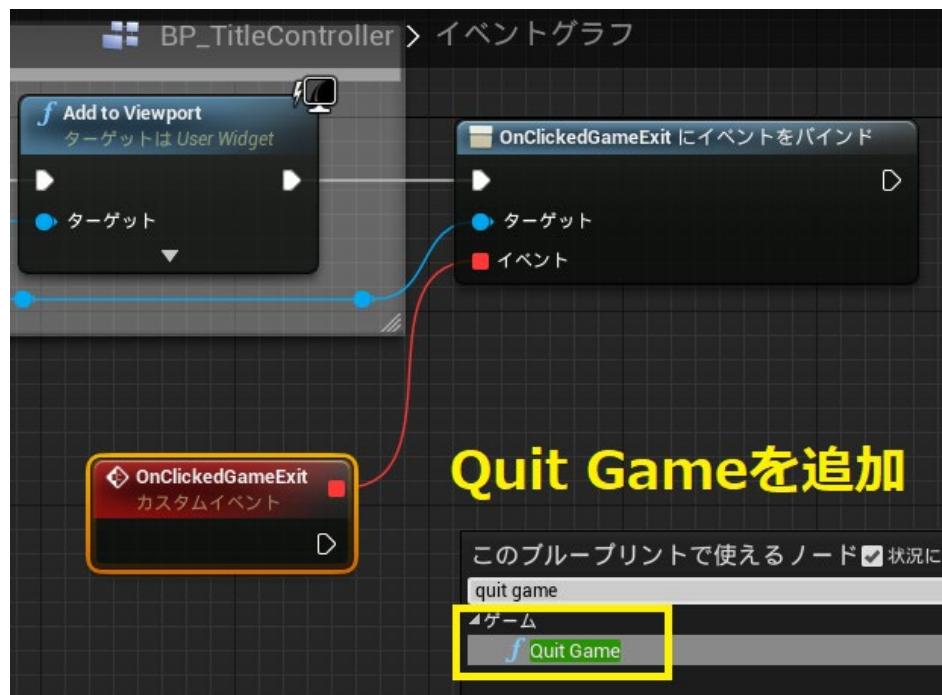
OnClickedGameExitにイベントをバインドし、ゲームを終了させる 1



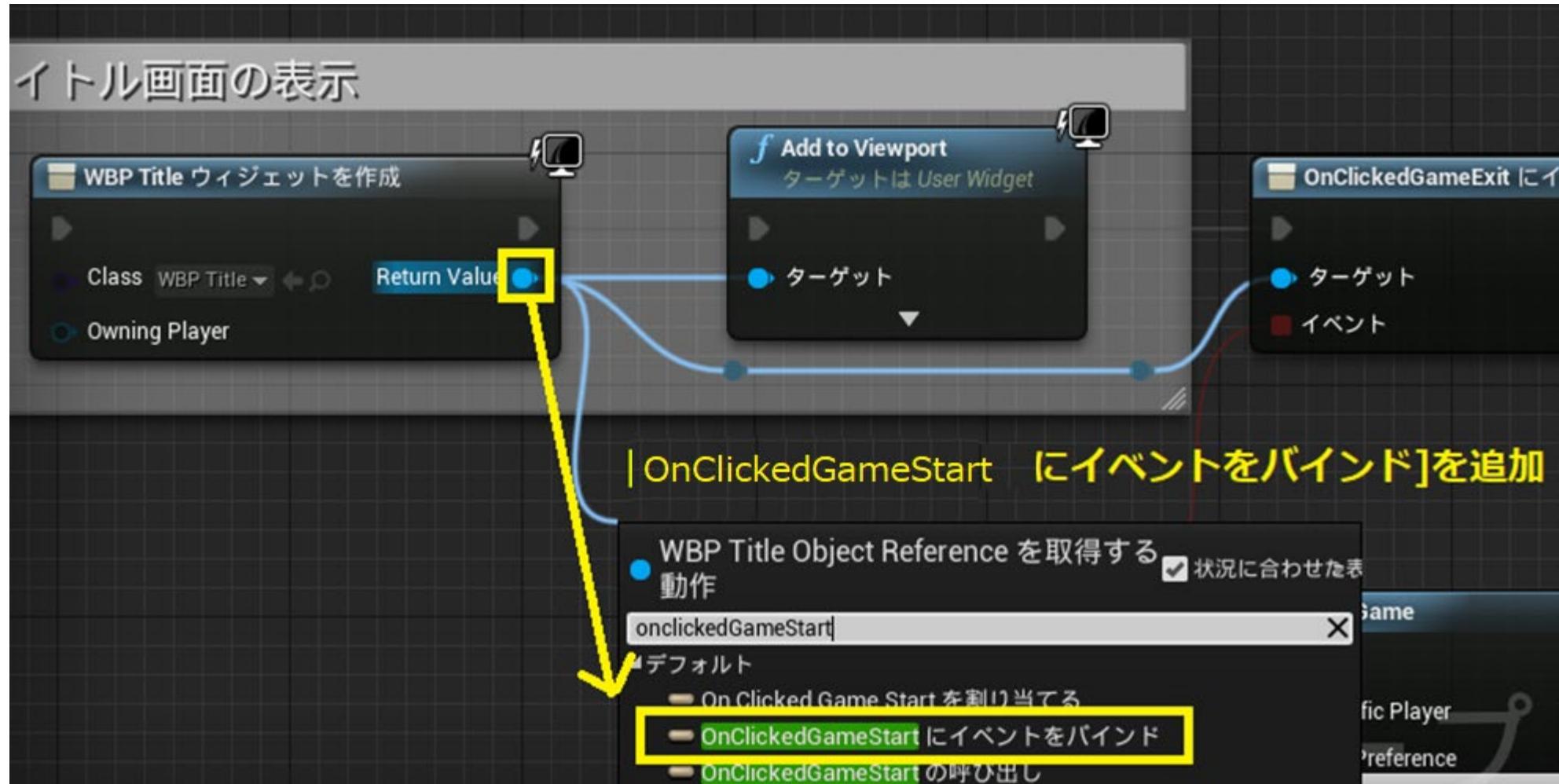
OnClickedGameExitにイベントをバインドし、ゲームを終了させる 2



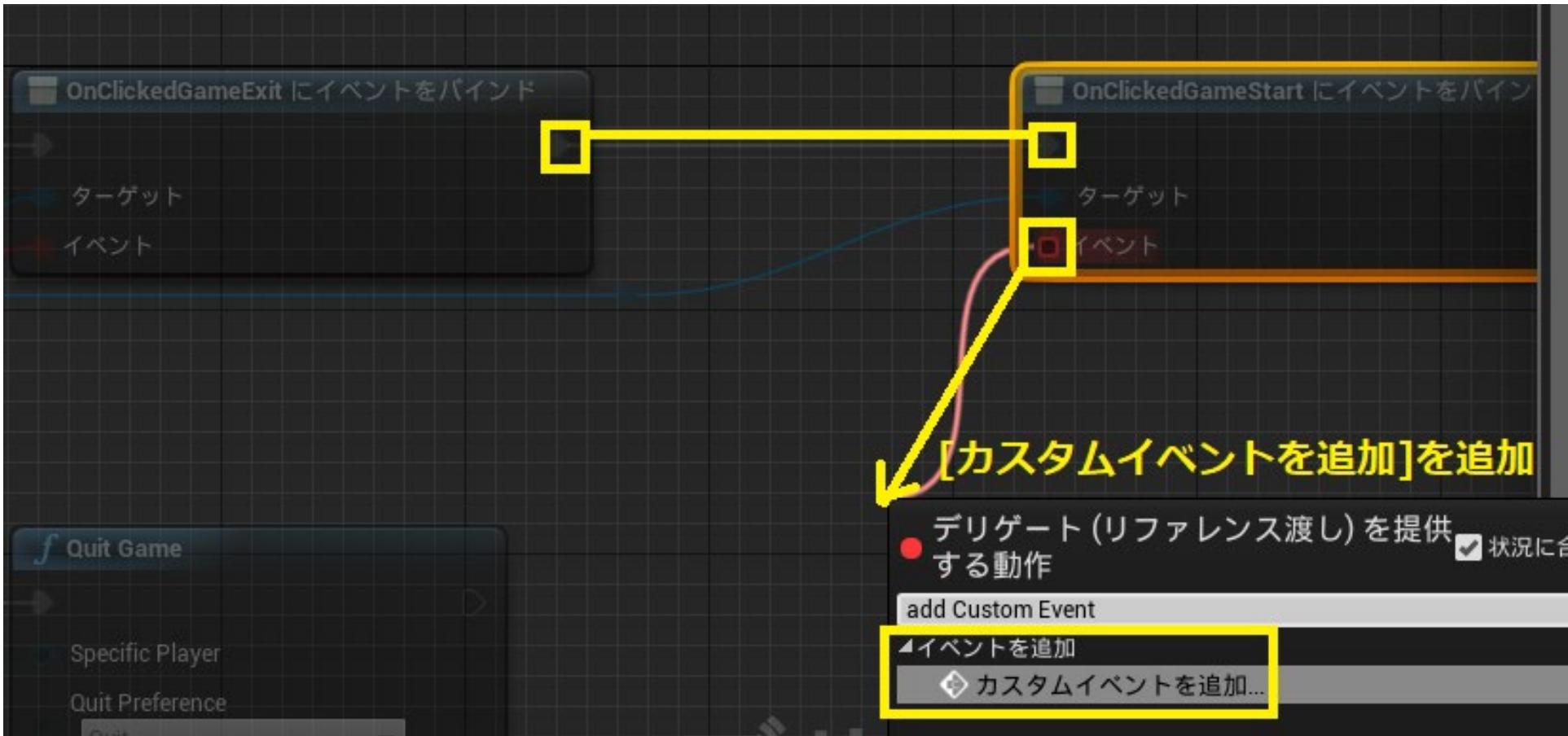
OnClickedGameExitにイベントをバインドし、ゲームを終了させる 3



OnClickedGameStartにイベントをバインドし、 レベルGameに遷移する処理を実装する 1



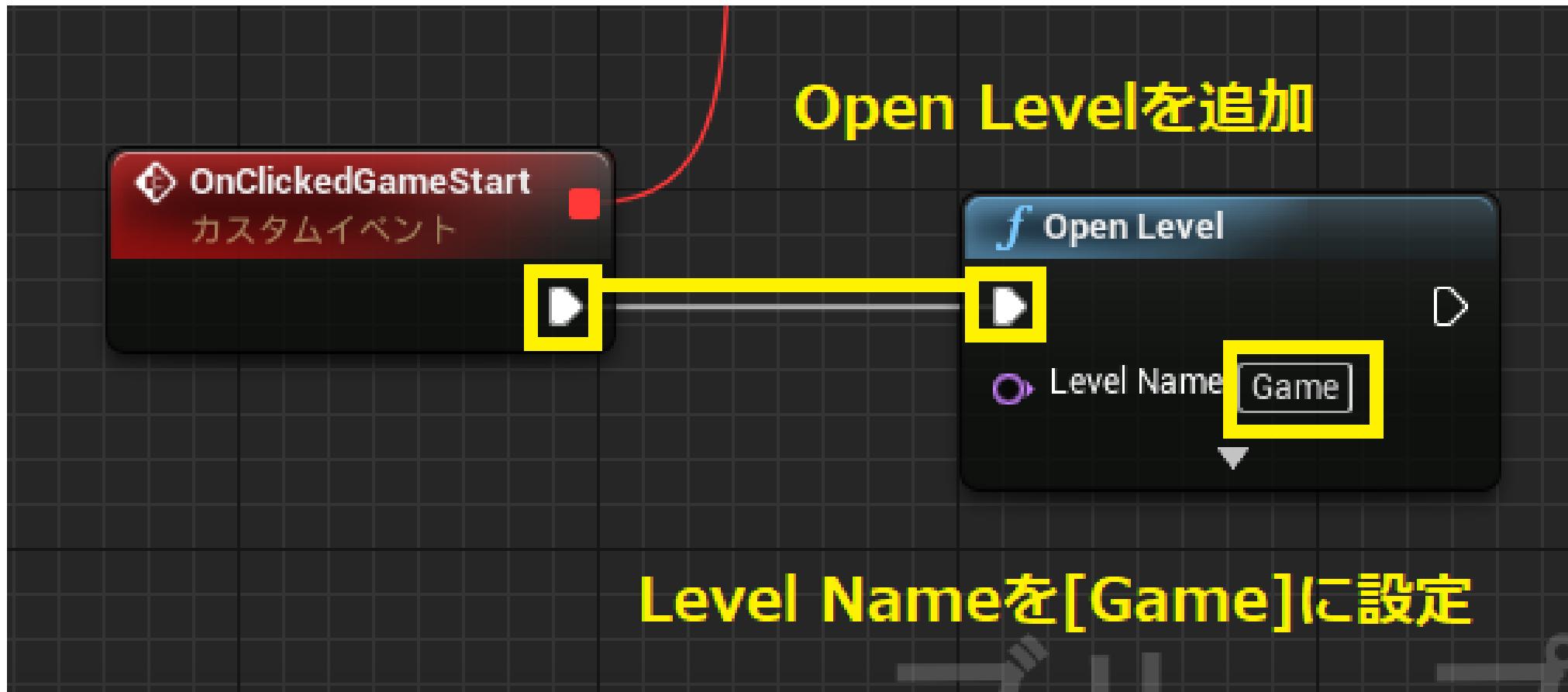
OnClickedGameStartにイベントをバインドし、 レベルGameに遷移する処理を実装する 2



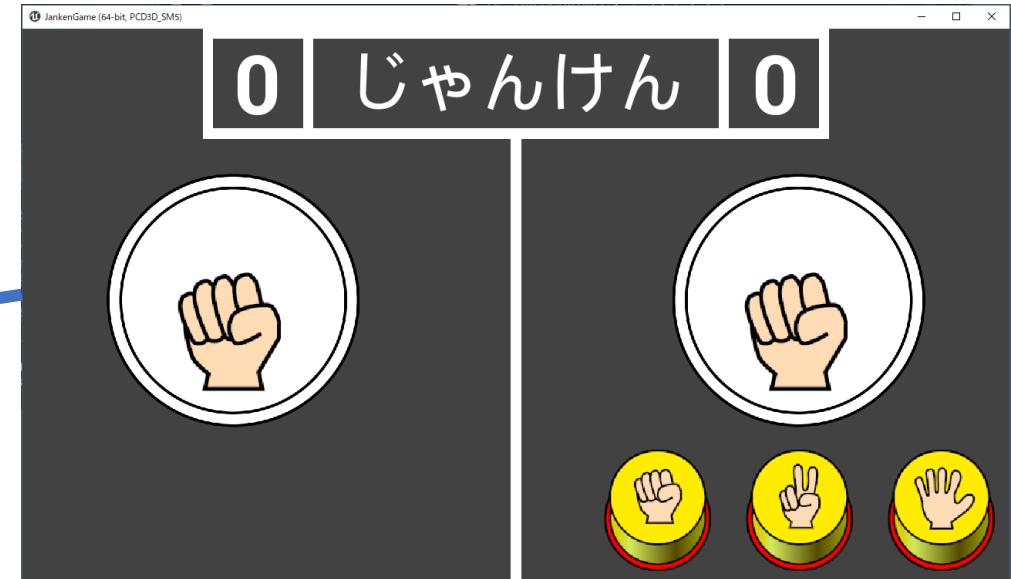
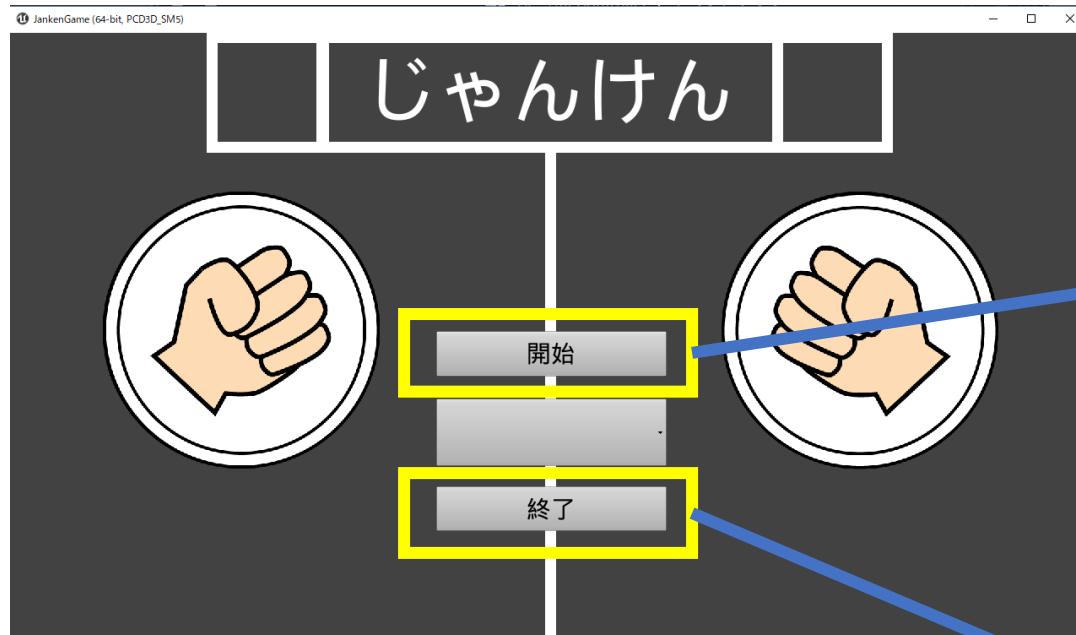
OnClickedGameStartにイベントをバインドし、
レベルGameに遷移する処理を実装する 3



OnClickedGameStartにイベントをバインドし、
レベルGameに遷移する処理を実装する 4



プレイして確認する



ゲーム終了

11. タイトルを表示するUI作成

11.1 新規レベルTitleを作成

11.2 WBP_Titleの作成

11.3 ウィジェットの配置

11.4 ブループリント : BP_TitleControllerを作成

11.5 WBP_Titleにボタンのイベントディスパッチャを追加

11.6 BP_TitleControllerにボタンのイベントをバインド処理を実装

11.7 Loading画面 WBP>Loadingを作成

11.8 レベル遷移前にLoading画面を表示

11.9 WBP_GameResultのタイトルボタンからTitleレベルに遷移

ウィジェットブループリント：WBP_Loadingを作成する

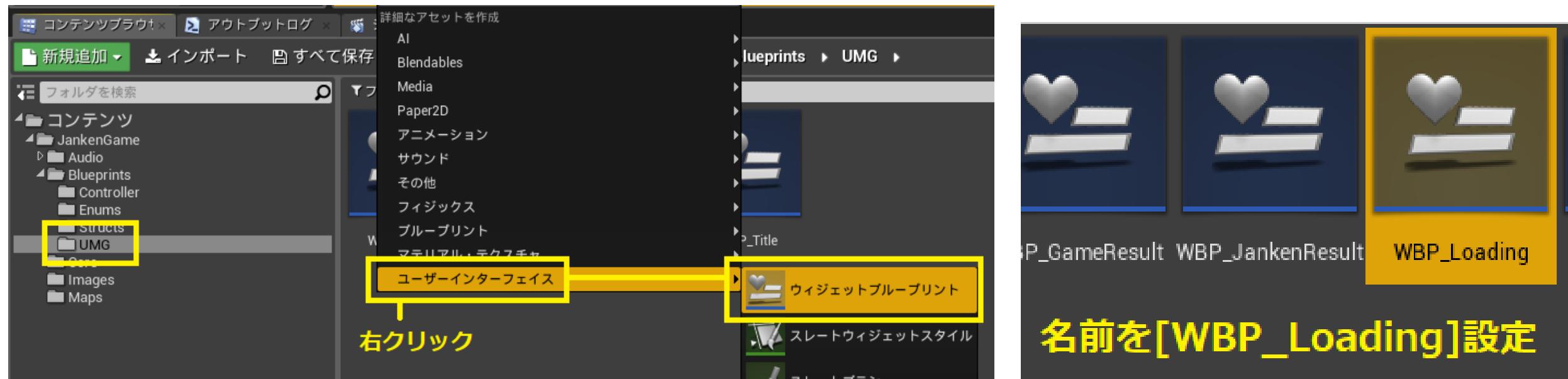


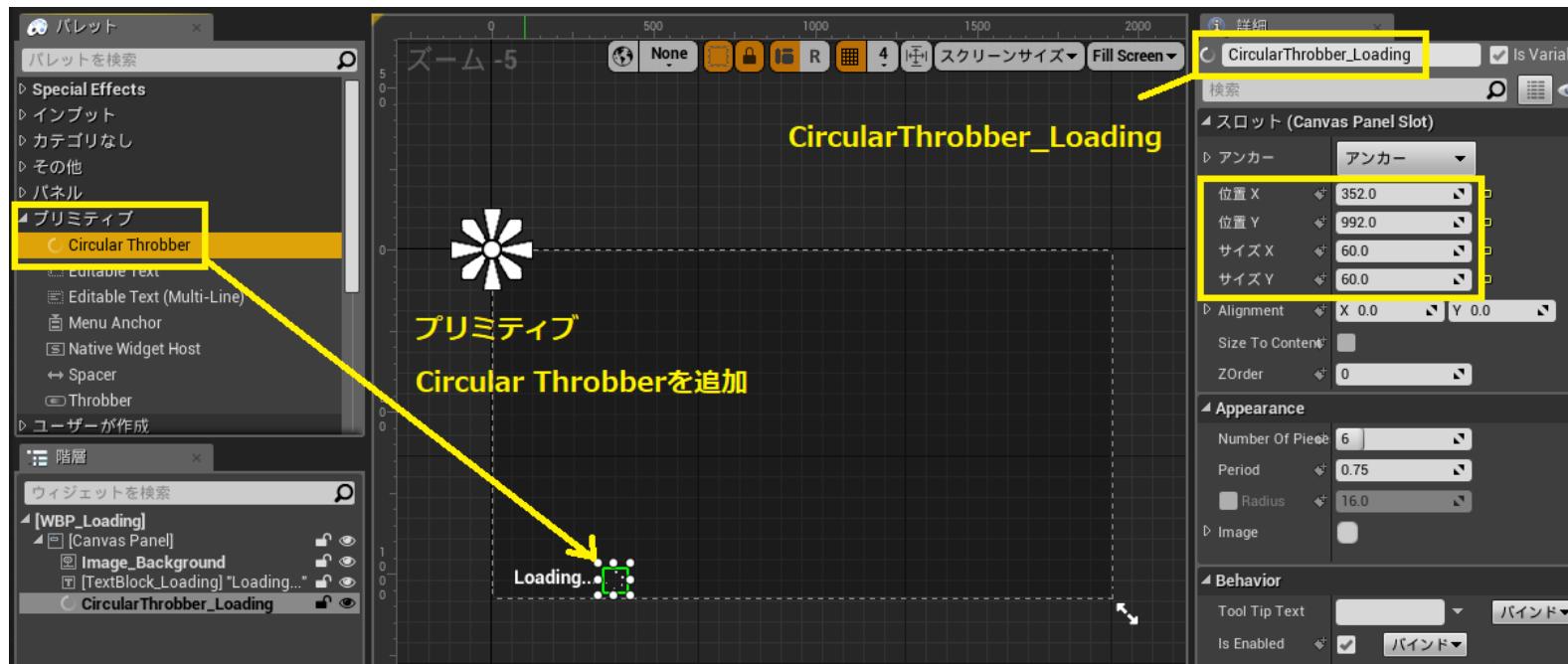
Image:Image_Backgroundを追加する

The screenshot shows the Construct 3 editor interface with the following components visible:

- Palette (Left):** Shows various UI elements like Border, Button, and Image.
- Canvas Panel (Center):** A green dashed rectangle representing the canvas panel.
- Properties Panel (Right):** Displays properties for the selected object "Image_Background".
 - Slot (Canvas Panel Slot):** Anchored at the center (0.0, 0.0) with no offsets.
 - Appearance:** Brush color is black (R:0.0 G:0.0 B:0.0 A:0.3).
- Details Panel (Top Right):** Shows the object's name "Image_Background" and its variable status.
- Properties Table (Bottom Right):** A summary of the object's properties.

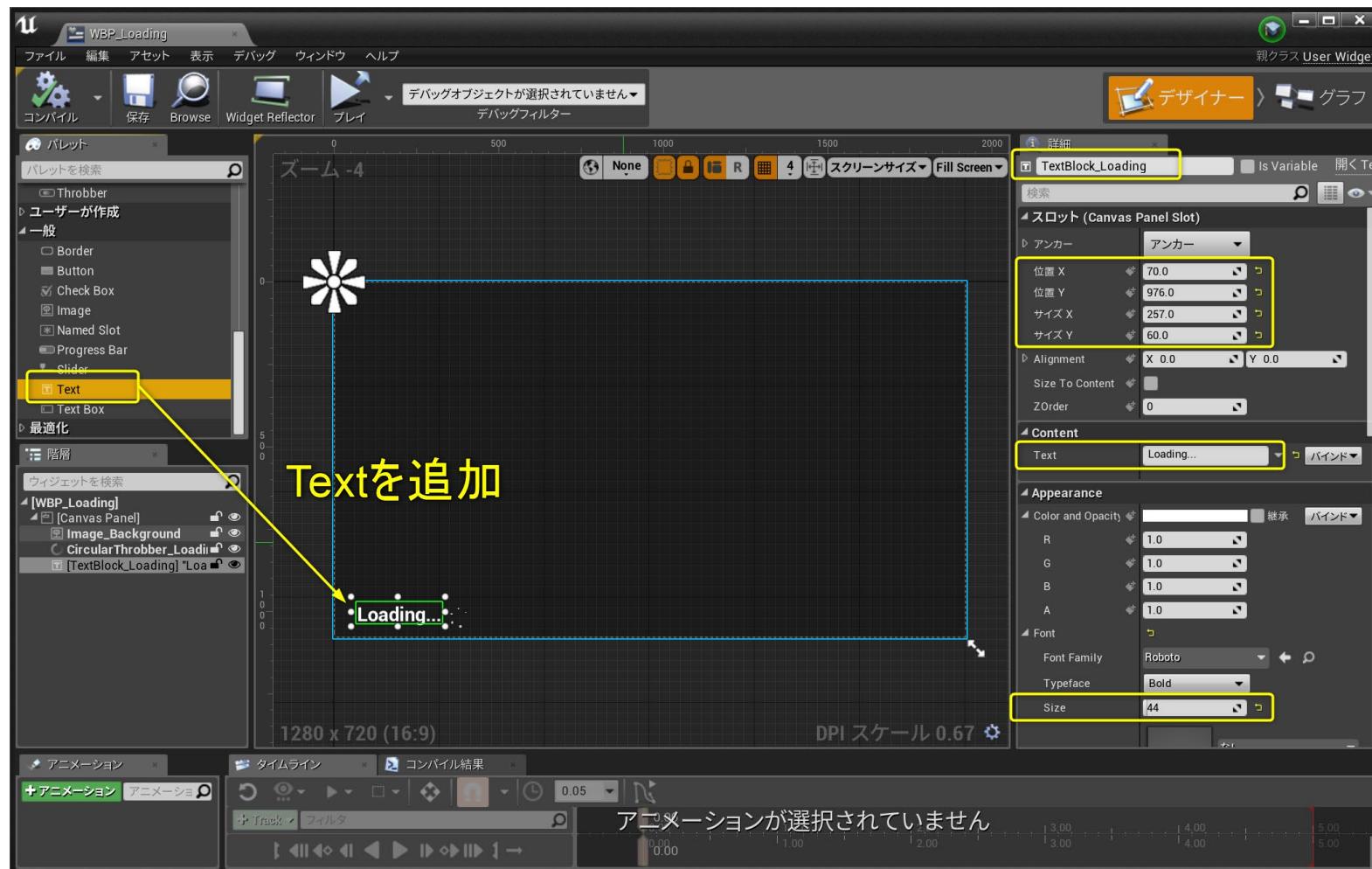
プロパティ	値
変数名	Image_Background
左オフセット	0.0
上オフセット	0.0
右オフセット	0.0
下オフセット	0.0
Color and Opacity	黒(R:0.0 G:0.0 B:0.0 A:0.3)

ClrcularThrobber:CircularThrobber_Loadingを追加する



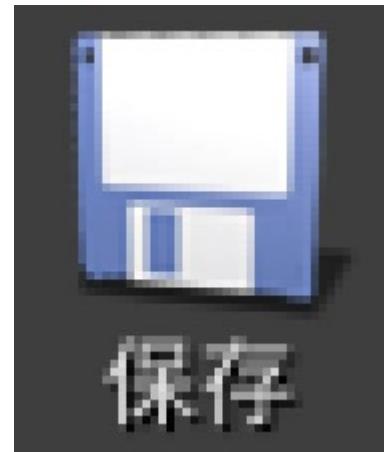
プロパティ	値
変数名	CircularThrobber_Loading
位置X	352.0
位置Y	992.0
サイズX	60.0
サイズY	60.0

Text:TextBlock_Loadingを追加する



プロパティ	値
変数名	TextBlock_Loading
位置X	70.0
位置Y	976.0
サイズX	257.0
サイズY	60.0
Text	Loading...
Font > Size	44

コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

11. タイトルを表示するUI作成

11.1 新規レベルTitleを作成

11.2 WBP_Titleの作成

11.3 ウィジェットの配置

11.4 ブループリント : BP_TitleControllerを作成

11.5 WBP_Titleにボタンのイベントディスパッチャを追加

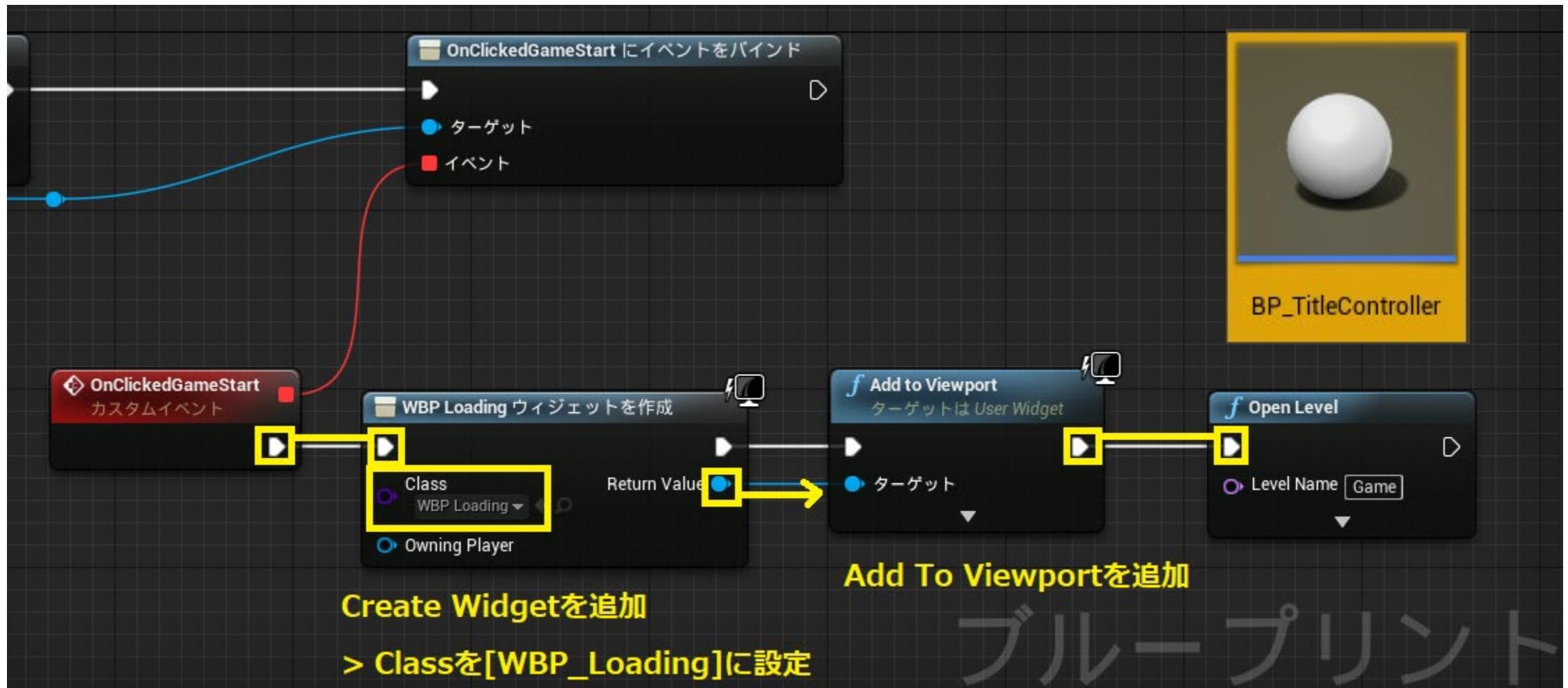
11.6 BP_TitleControllerにボタンのイベントをバインド処理を実装

11.7 Loading画面 WBP>Loadingを作成

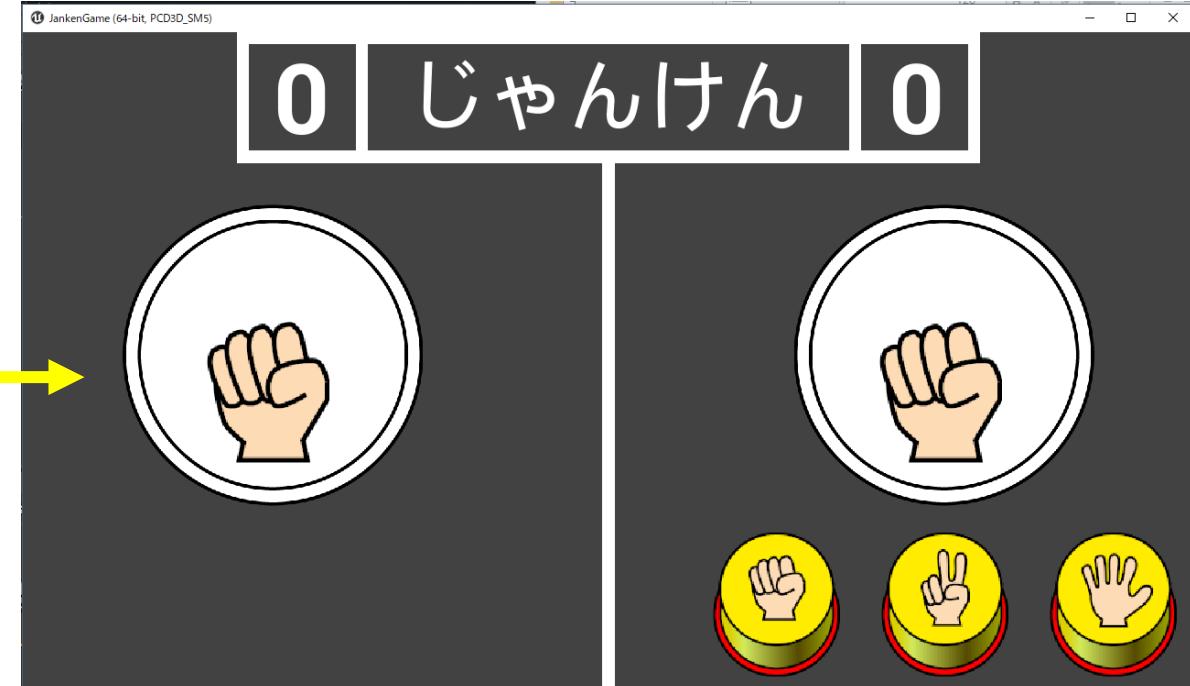
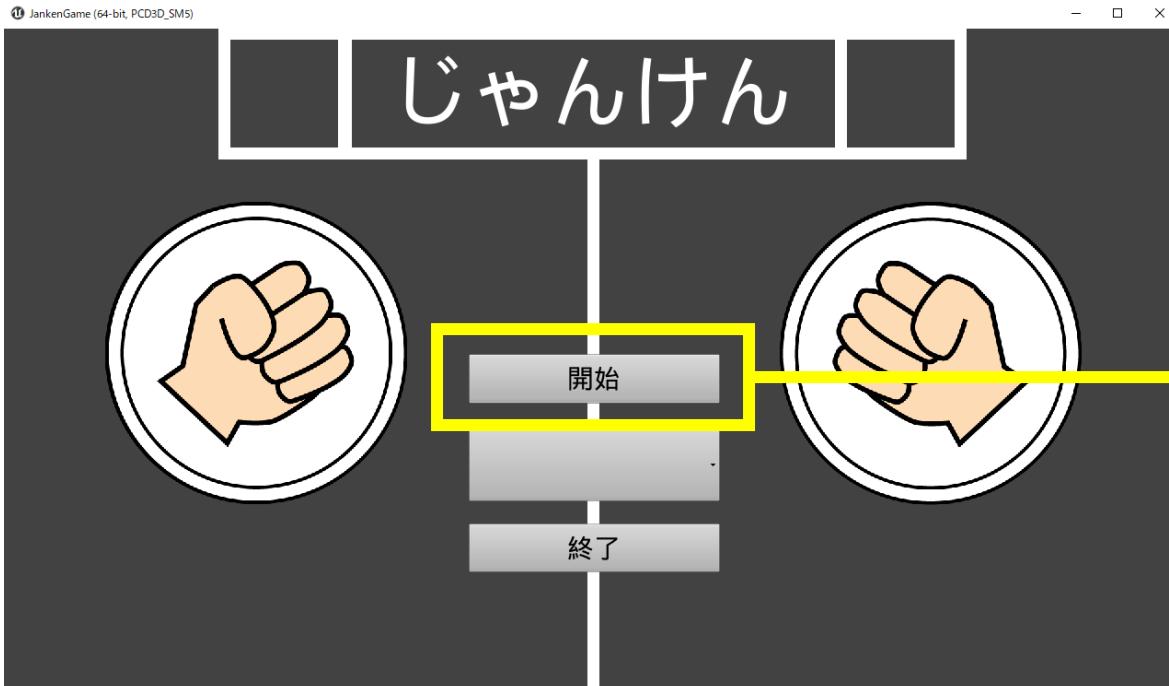
11.8 レベル遷移前にLoading画面を表示

11.9 WBP_GameResultのタイトルボタンからTitleレベルに遷移

レベル遷移前にLoading画面を表示する 1

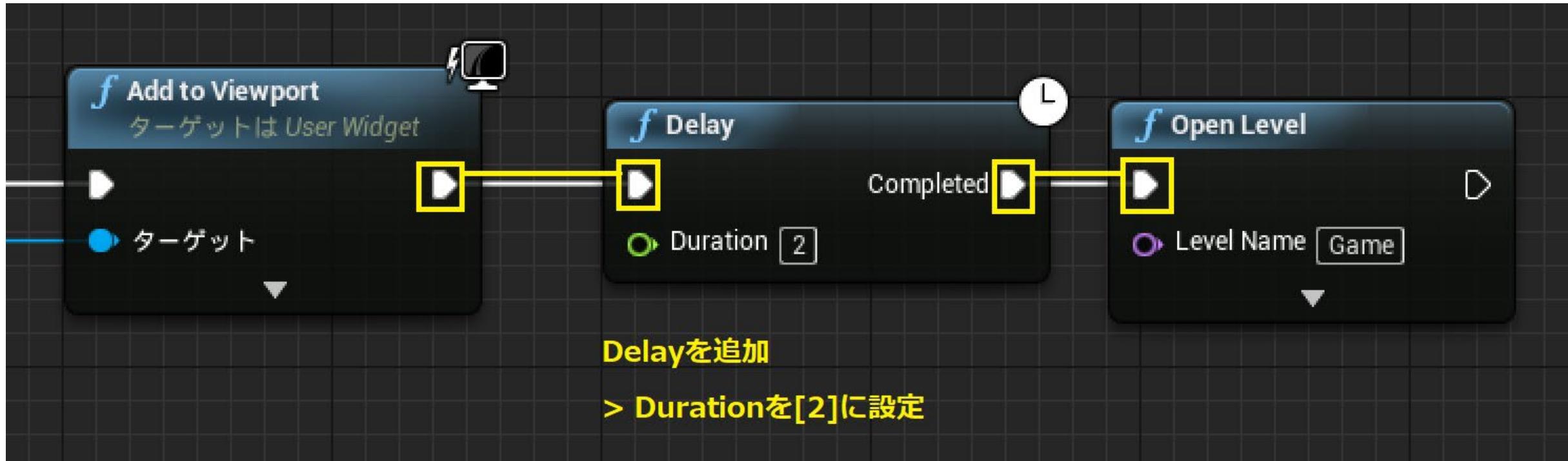


プレイして確認する



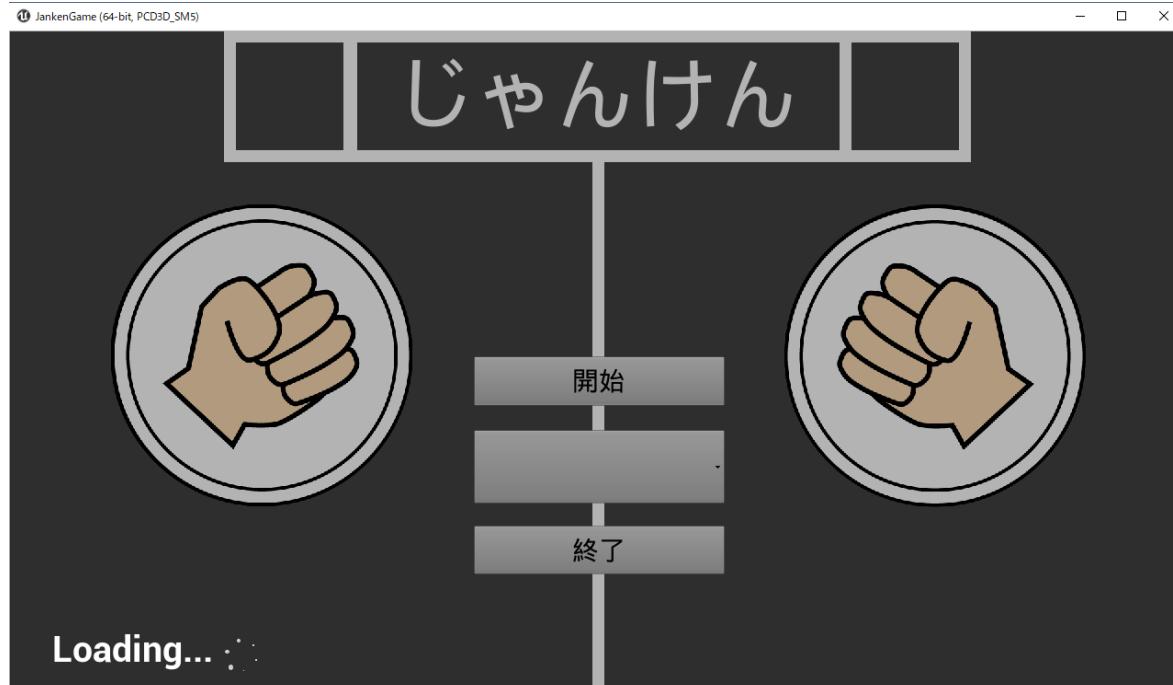
じゃんけんゲームの規模だと一瞬でレベル遷移してしまう

レベル遷移前にLoading画面を表示する 2

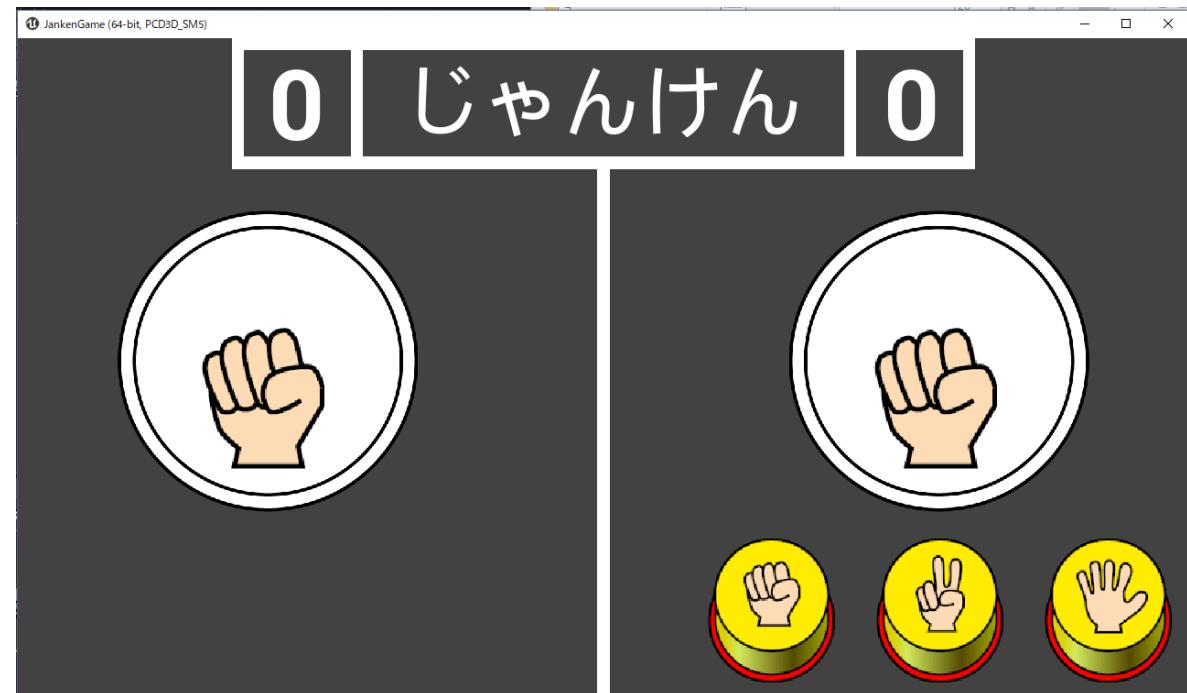


Delayノードを追加して、2秒間ローディング画面を表示する
(ゲームを作成する際には強引に表示する必要はないです)

プレイして確認する



開始ボタンをクリックしてからローディング画面が表示され、
Gameレベルに遷移する



コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

11. タイトルを表示するUI作成

11.1 新規レベルTitleを作成

11.2 WBP_Titleの作成

11.3 ウィジェットの配置

11.4 ブループリント : BP_TitleControllerを作成

11.5 WBP_Titleにボタンのイベントディスパッチャを追加

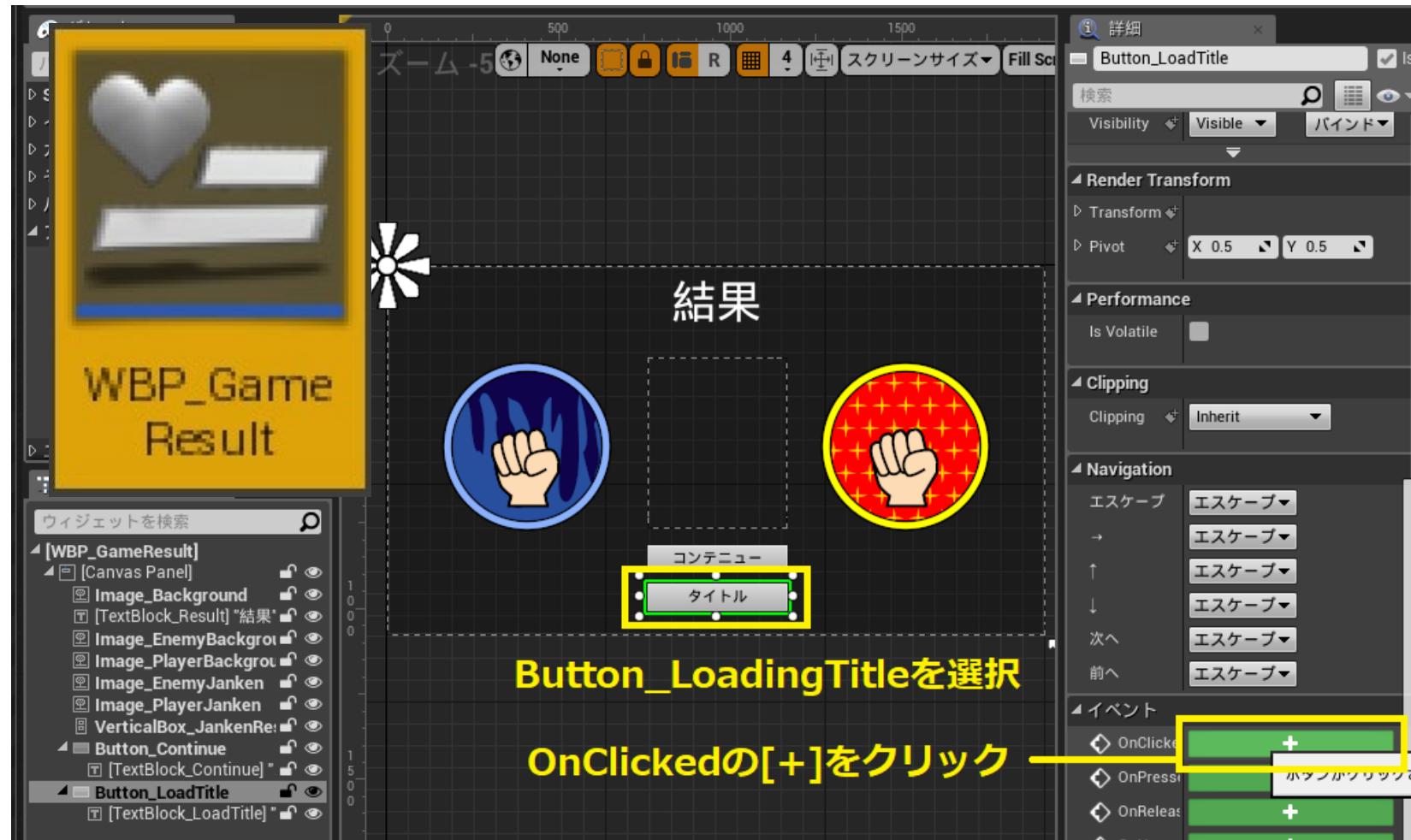
11.6 BP_TitleControllerにボタンのイベントをバインド処理を実装

11.7 Loading画面 WBP>Loadingを作成

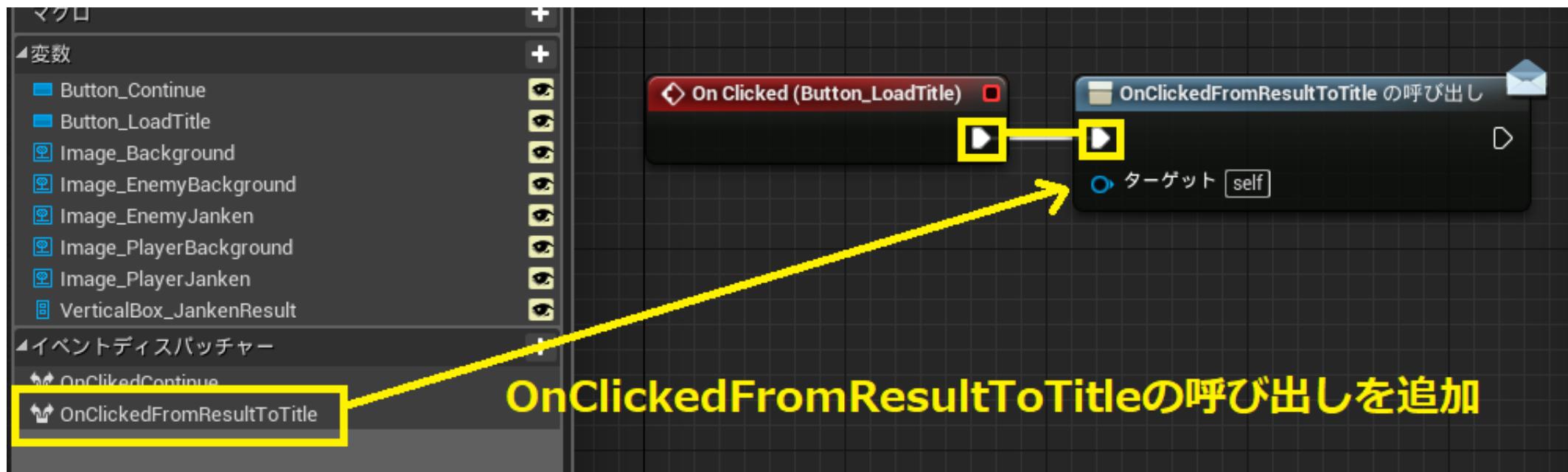
11.8 レベル遷移前にLoading画面を表示

11.9 WBP_GameResultのタイトルボタンからTitleレベルに遷移

WBP_GameResultのButton_LoadTitleにOnClickedイベントを追加する 1



WBP_GameResultのButton_LoadingTitleにOnClickedイベントを追加する
2
イベントディスパッチャー:OnClickedFromResultToTitleを追加する

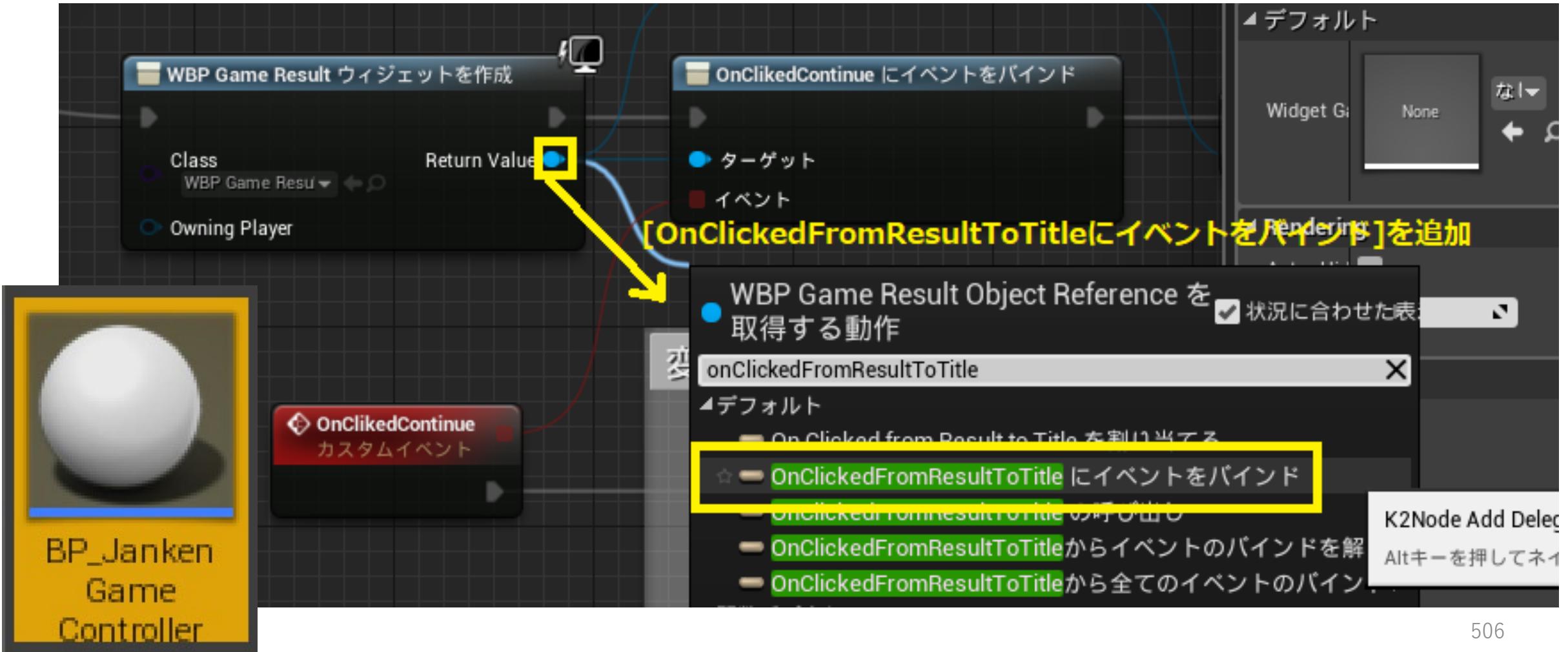


コンパイル > 保存

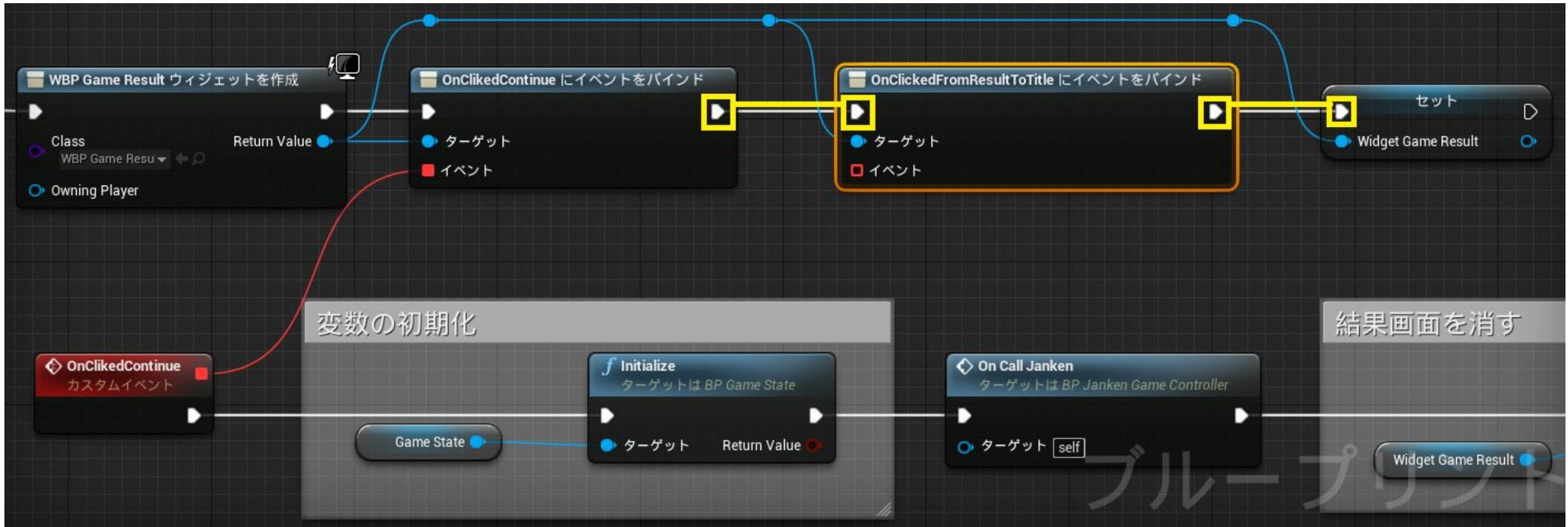


この後に別のブループリントを編集するため、コンパイル > 保存

WBP_GameResultの OnClickedFromResultToTitleにイベントをバインドを追加する 1



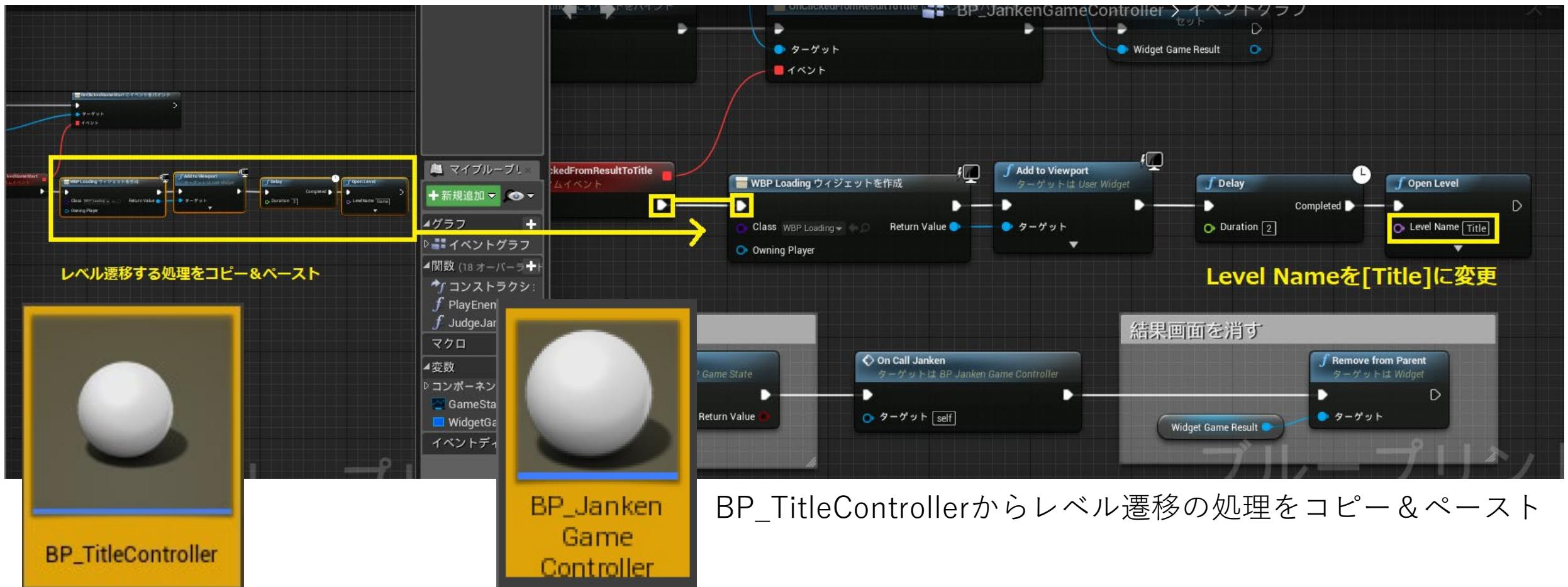
WBP_GameResultの OnClickedFromResultToTitleにイベントをバインドを追加する 2



WBP_GameResultの OnClickedFromResultToTitleにイベントをバインドを追加する 3



WBP_GameResultの OnClickedFromResultToTitleにイベントをバインドを追加する 3

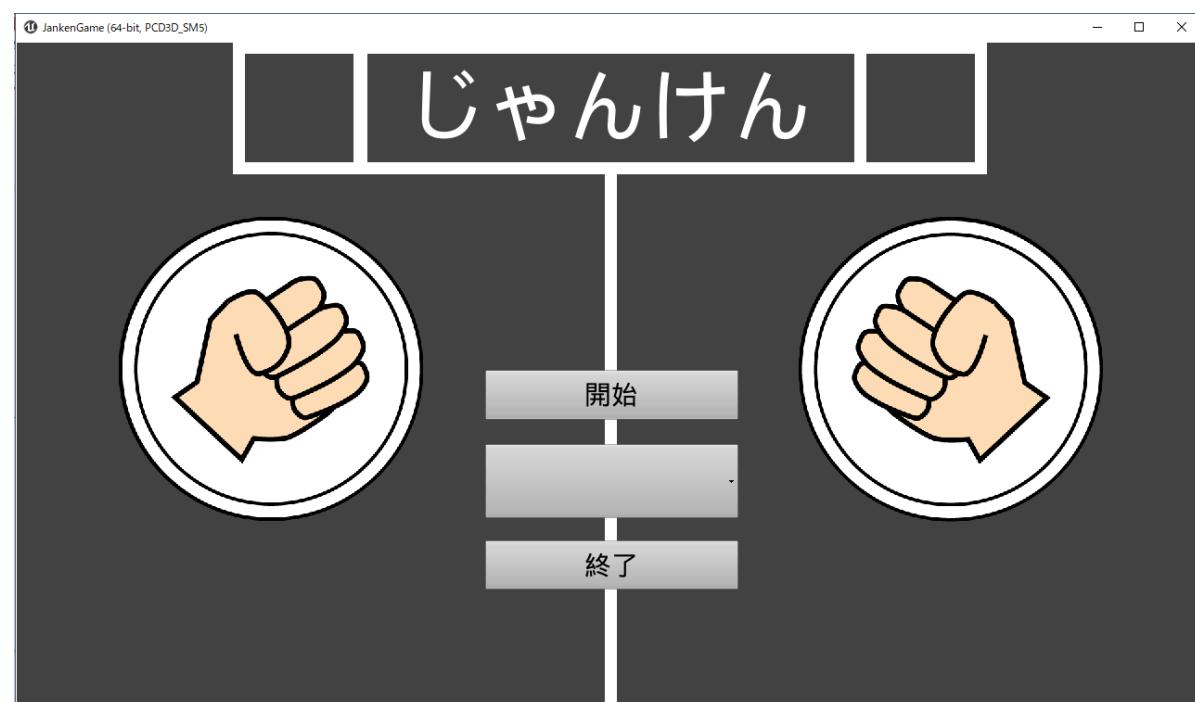


コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

プレイして確認する



12. メニュー画面の作成

12. メニュー画面の作成

12.1 WBP_Menuの作成

12.2 WBP_Menuを表示

12.3 WBP_Menuのボタンからイベントディスパッチャを呼び出す

12.4 WBP_Menuのボタンイベントをバインドする

 12.4.1 再開ボタンのイベントにバインド

 12.4.2 タイトルボタンのイベントにバインド

 12.4.3 終了ボタンのイベントにバインド

12. メニュー画面の作成

12.1 WBP_Menuの作成

12.2 WBP_Menuを表示

12.3 WBP_Menuのボタンからイベントディスパッチャを呼び出す

12.4 WBP_Menuのボタンイベントをバインドする

12.4.1 再開ボタンのイベントにバインド

12.4.2 タイトルボタンのイベントにバインド

12.4.3 終了ボタンのイベントにバインド

ウィジェットブループリント:WBP_Menuを作成する

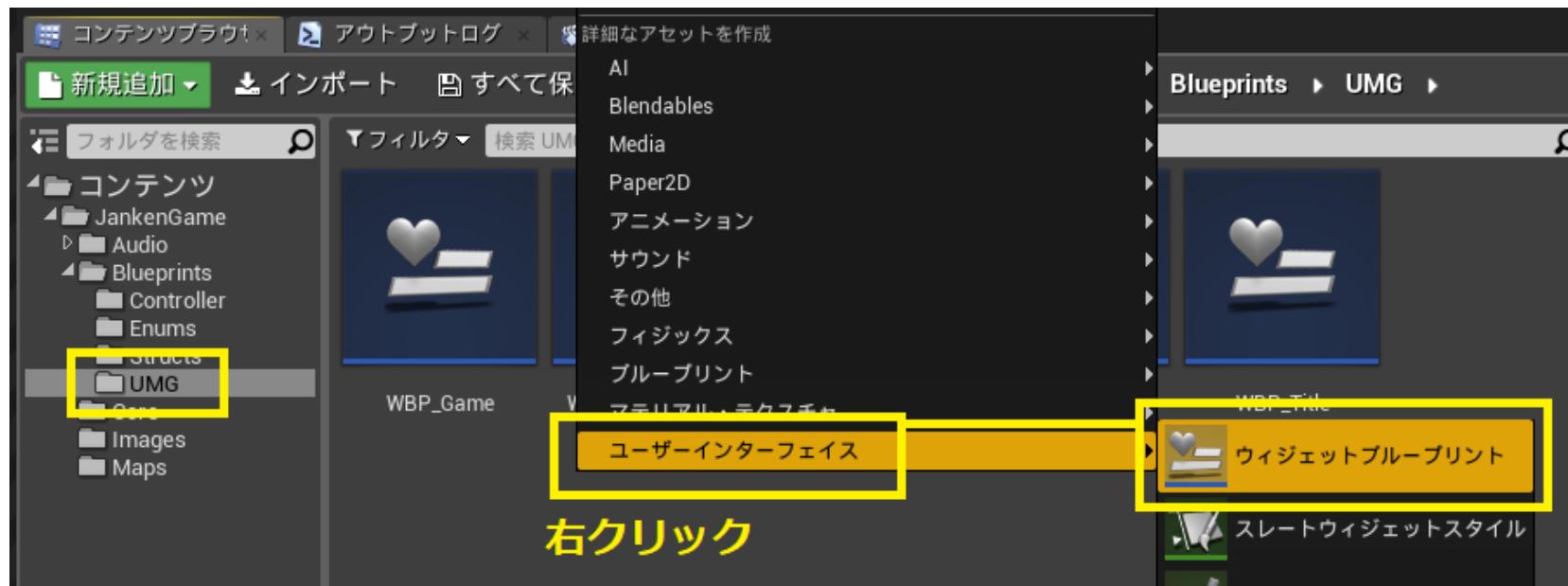


Image:Image_Backgroundを追加する

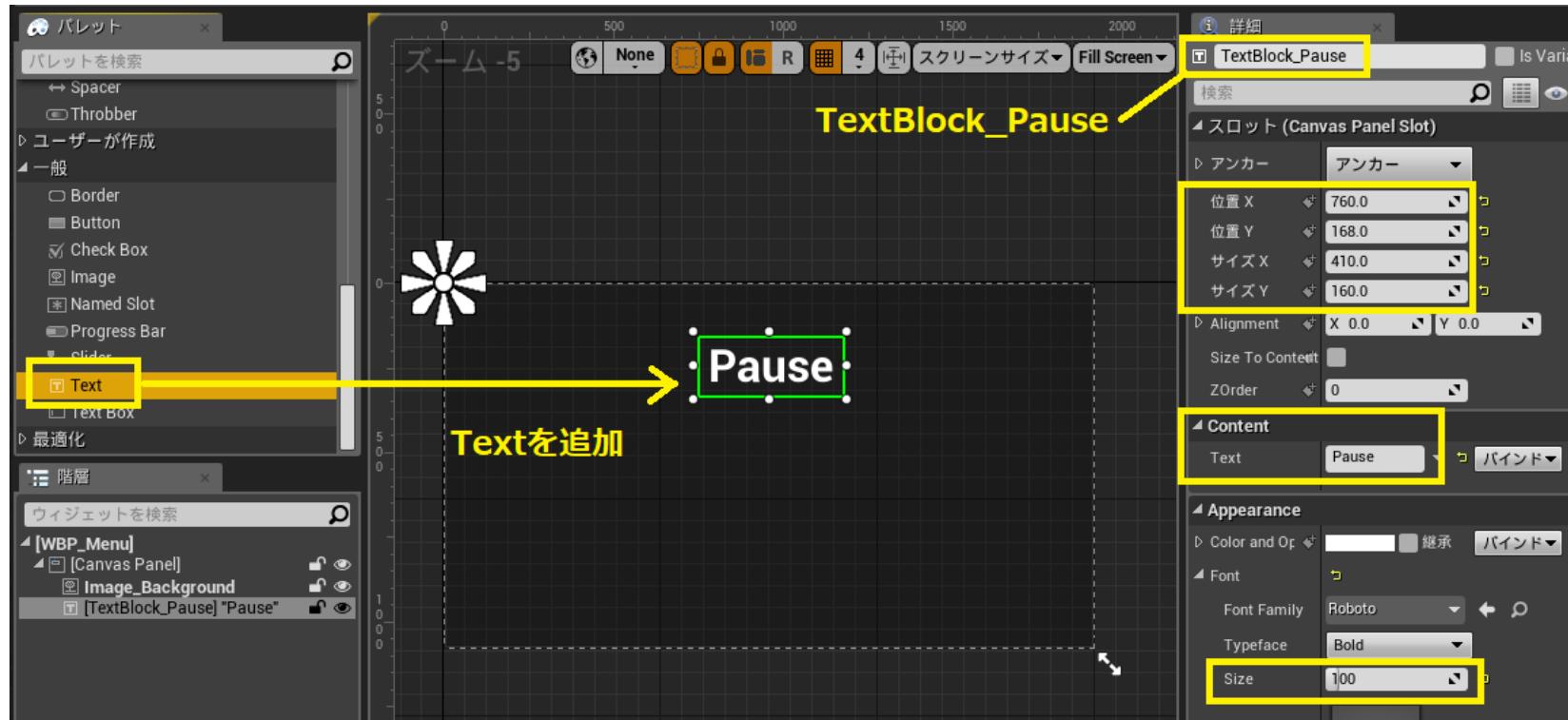
The screenshot shows the Construct 3 editor interface. On the left, the Palette panel has 'Image' selected. In the center, a new 'Image' component is being created on the canvas, with a dashed green outline and a bounding box. A yellow arrow points from the palette to the canvas with the text 'Imageを追加'. On the right, the Properties panel shows the component details:

- Image_Background** (highlighted)
- スロット (Canvas Panel Slot)**:
 - アンカー: アンカー (highlighted)
 - 左オフセット: 0.0 (highlighted)
 - 上オフセット: 0.0 (highlighted)
 - 右オフセット: 0.0 (highlighted)
 - 下オフセット: 0.0 (highlighted)
- Appearance**:
 - Brush: バインド (highlighted)
 - Color and Opacity:
 - R: 0.0 (highlighted)
 - G: 0.0 (highlighted)
 - B: 0.0 (highlighted)
 - A: 0.3 (highlighted)

Below the Properties panel, the **詳細 (Details)** panel shows the component's properties:

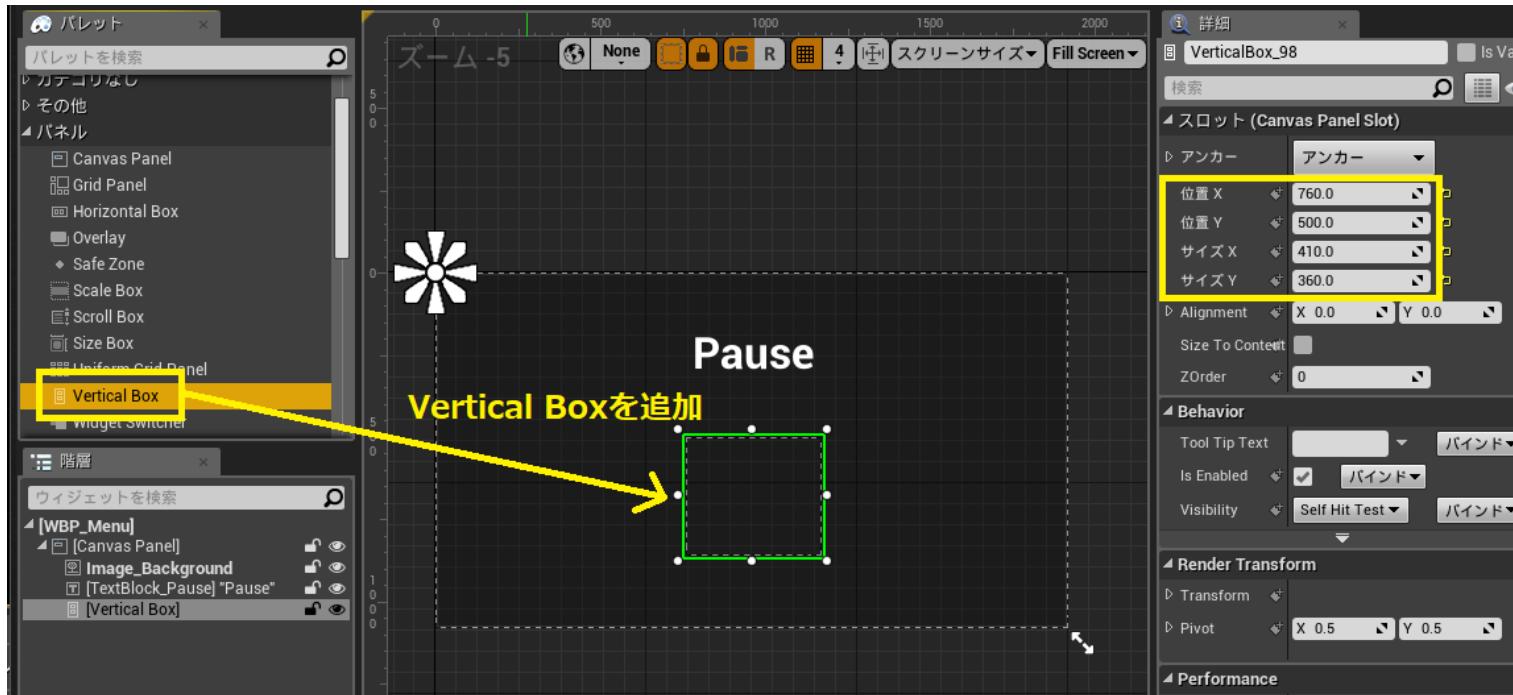
プロパティ (Property)	値 (Value)
変数名 (Variable Name)	Image_Background
左オフセット (Left Offset)	0.0
上オフセット (Top Offset)	0.0
右オフセット (Right Offset)	0.0
下オフセット (Bottom Offset)	0.0
Color and Opacity (Color and Opacity)	黒(R:0.0 G:0.0 B:0.0 A:0.3)

Text:TextBlock_Pauseを追加する



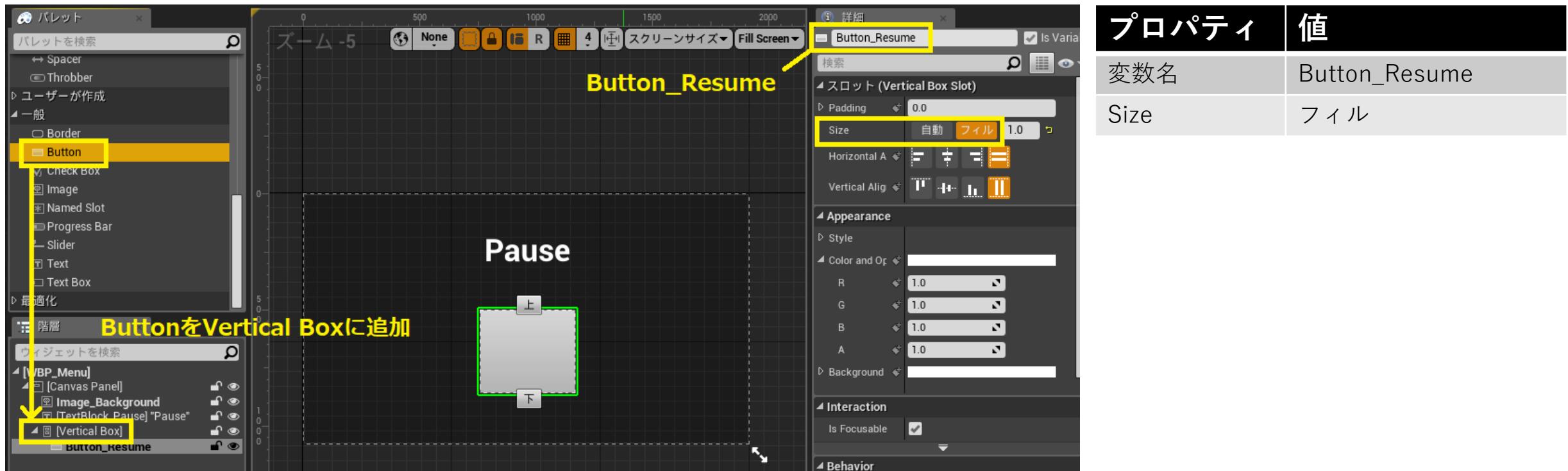
プロパティ	値
変数名	TextBlock_Pause
位置X	760.0
位置Y	168.0
サイズX	410.0
サイズY	160.0
Text	Pause
Font > Size	100
Justification	テキスト中央揃え

Vertical Boxを追加する

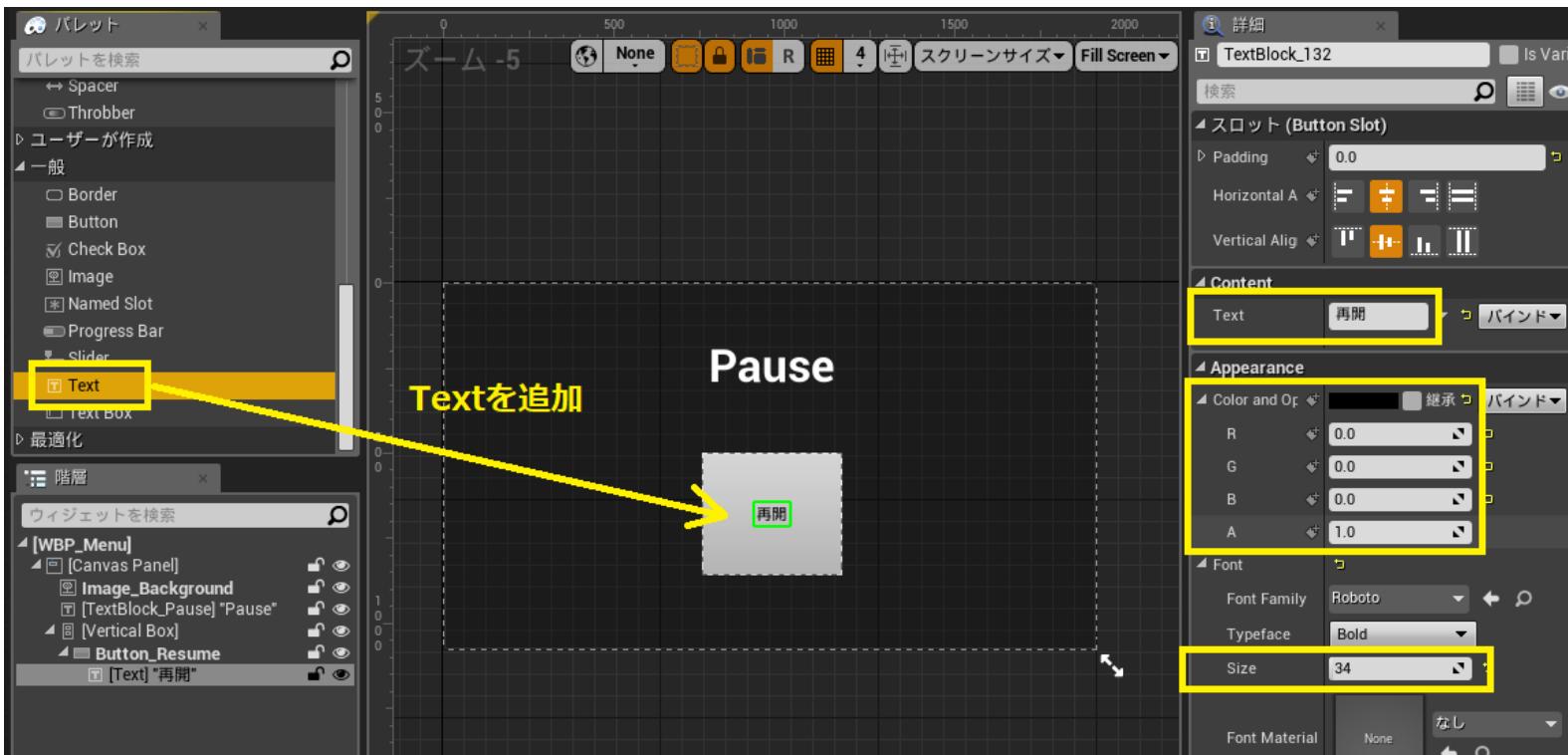


プロパティ	値
位置X	760.0
位置Y	500.0
サイズX	410.0
サイズY	360.0

Button:ButtonResumeをVertical Boxに追加する

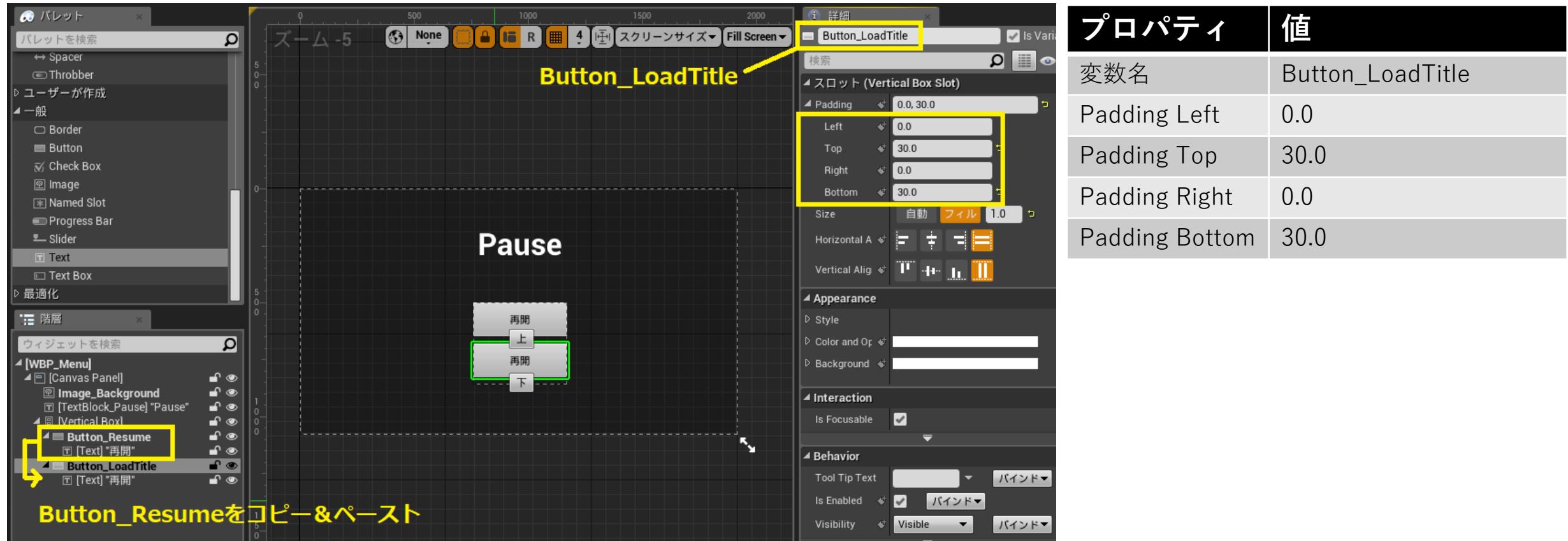


TextをButton_Resumeに追加する

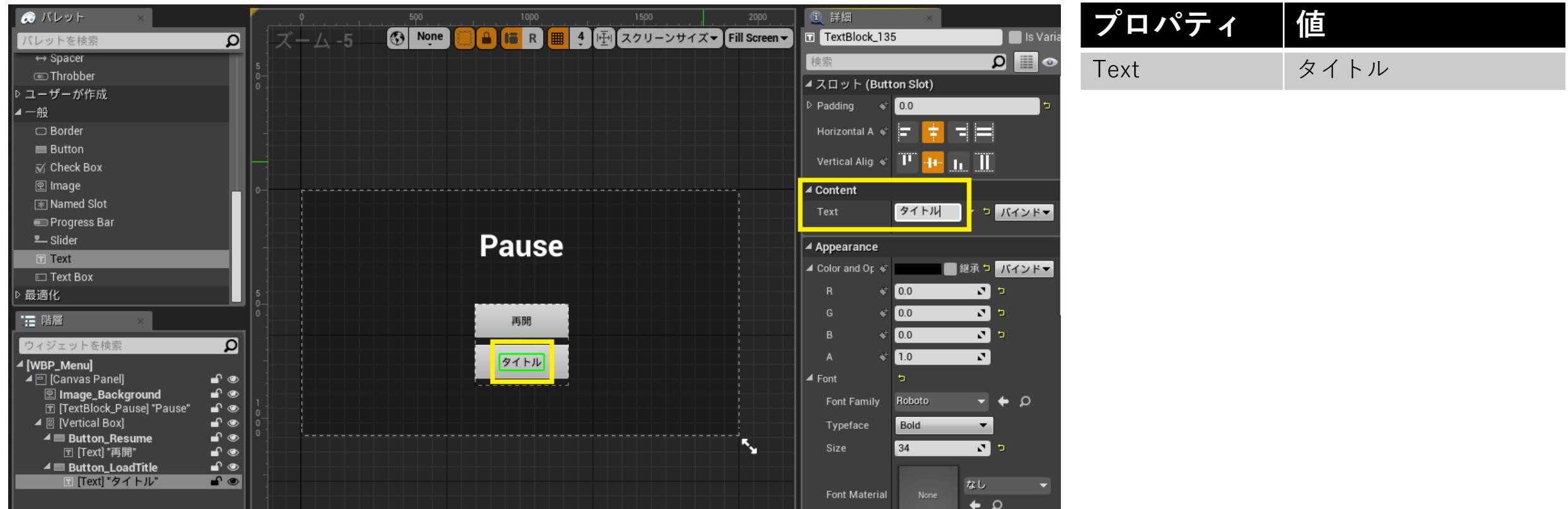


プロパティ	値
Text	再開
Color and Opacity	黒(R:0.0, G:0.0 B:0.0 A:1.0)
Font > Size	34

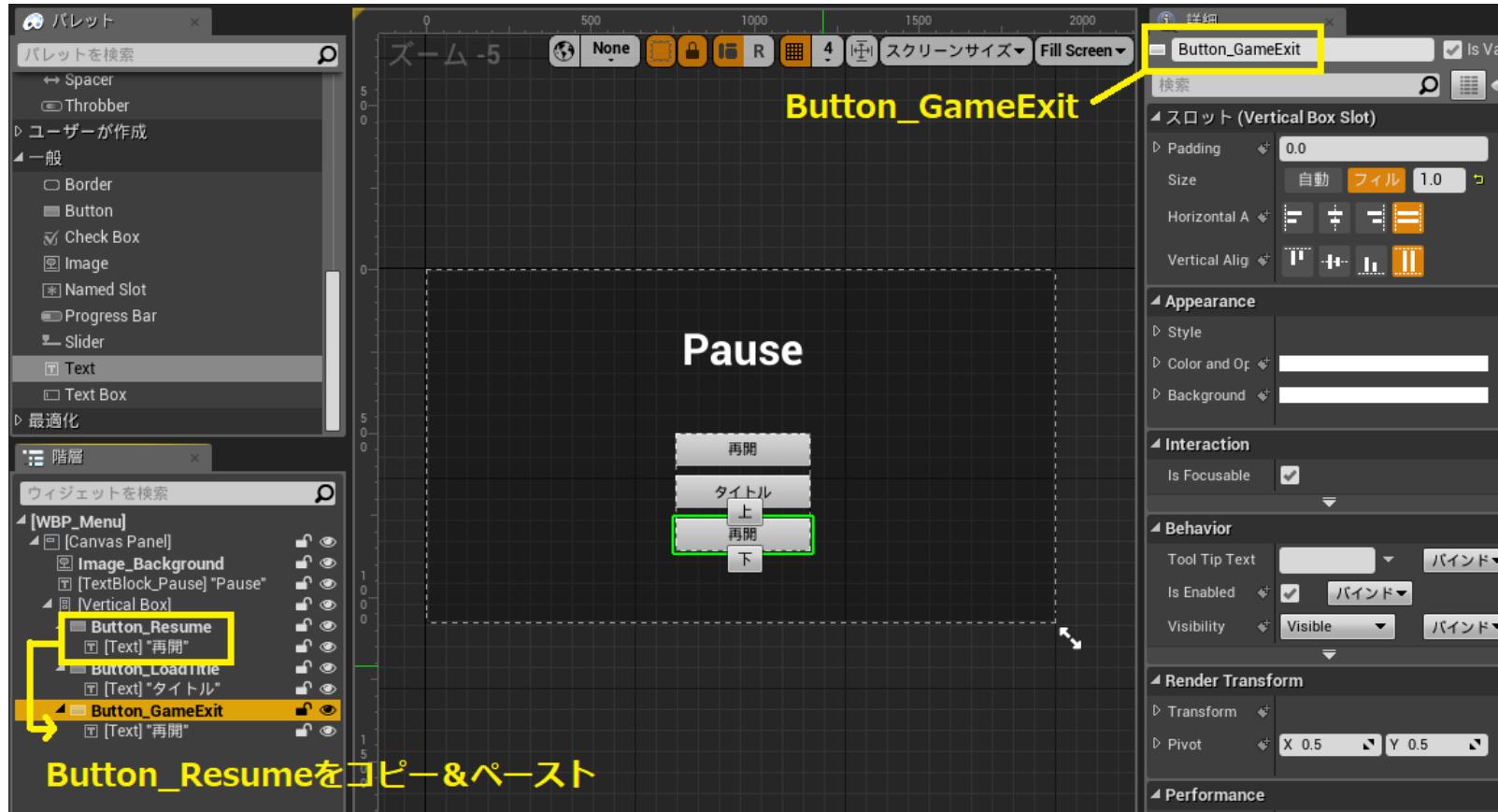
Button_Resumeを複製して、Button_LoadTitleを作成する



Button_LoadTitleのテキストを変更する



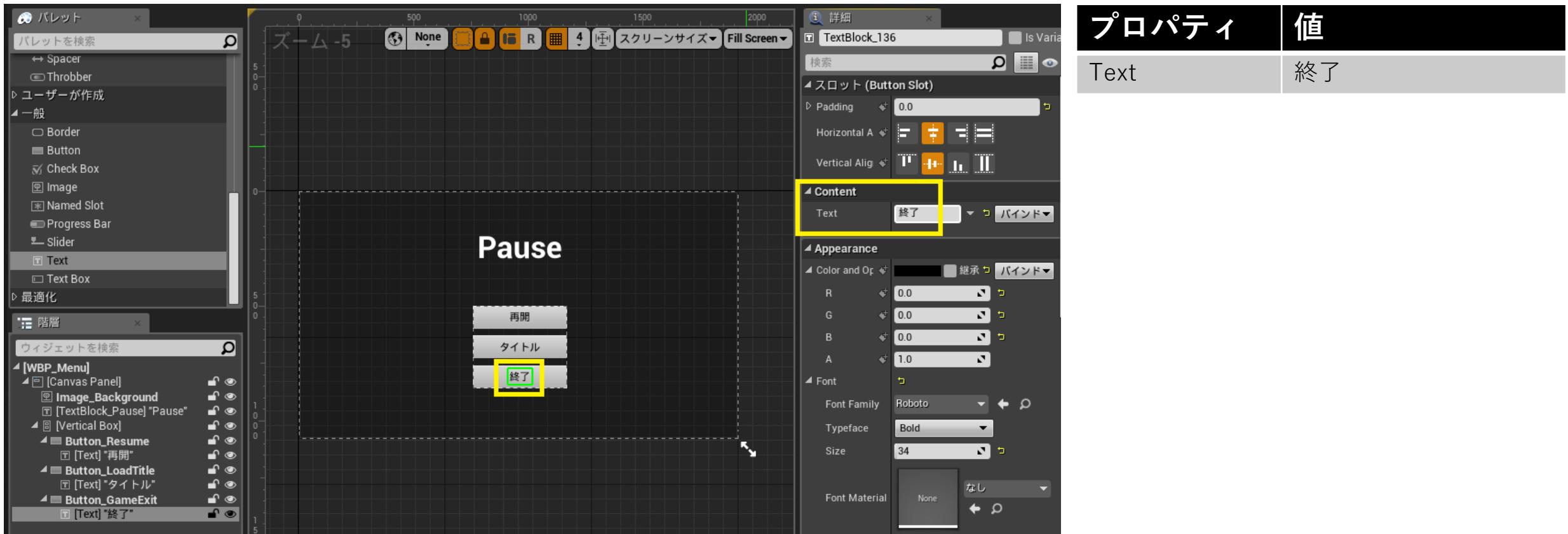
Button_Resumeを複製して、 Button_GameExitを作成する



Button_Resumeをコピー&ペースト

プロパティ	値
変数名	Button_GameExit

Button_GameExitのテキストを変更する



12. メニュー画面の作成

12.1 WBP_Menuの作成

12.2 WBP_Menuを表示

12.3 WBP_Menuのボタンからイベントディスパッチャを呼び出す

12.4 WBP_Menuのボタンイベントをバインドする

 12.4.1 再開ボタンのイベントにバインド

 12.4.2 タイトルボタンのイベントにバインド

 12.4.3 終了ボタンのイベントにバインド

BP_JankenGameControllerに キーボードイベント：スペースバーを追加する 1



BP_JankenGameControllerに キーボードイベント：スペースバーを追加する 2



コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

プレイして確認する



12. メニュー画面の作成

12.1 WBP_Menuの作成

12.2 WBP_Menuを表示

12.3 WBP_Menuのボタンからイベントディスパッチャを呼び出す

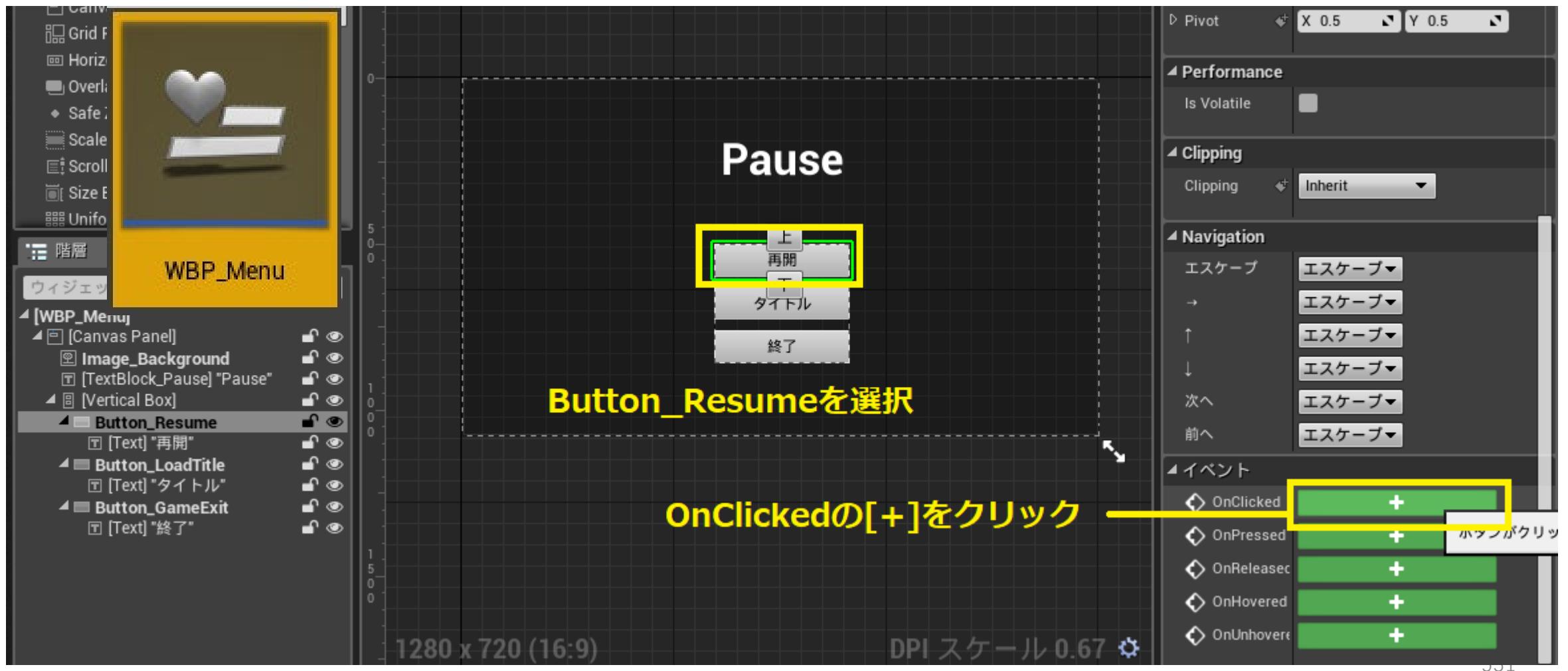
12.4 WBP_Menuのボタンイベントをバインドする

 12.4.1 再開ボタンのイベントにバインド

 12.4.2 タイトルボタンのイベントにバインド

 12.4.3 終了ボタンのイベントにバインド

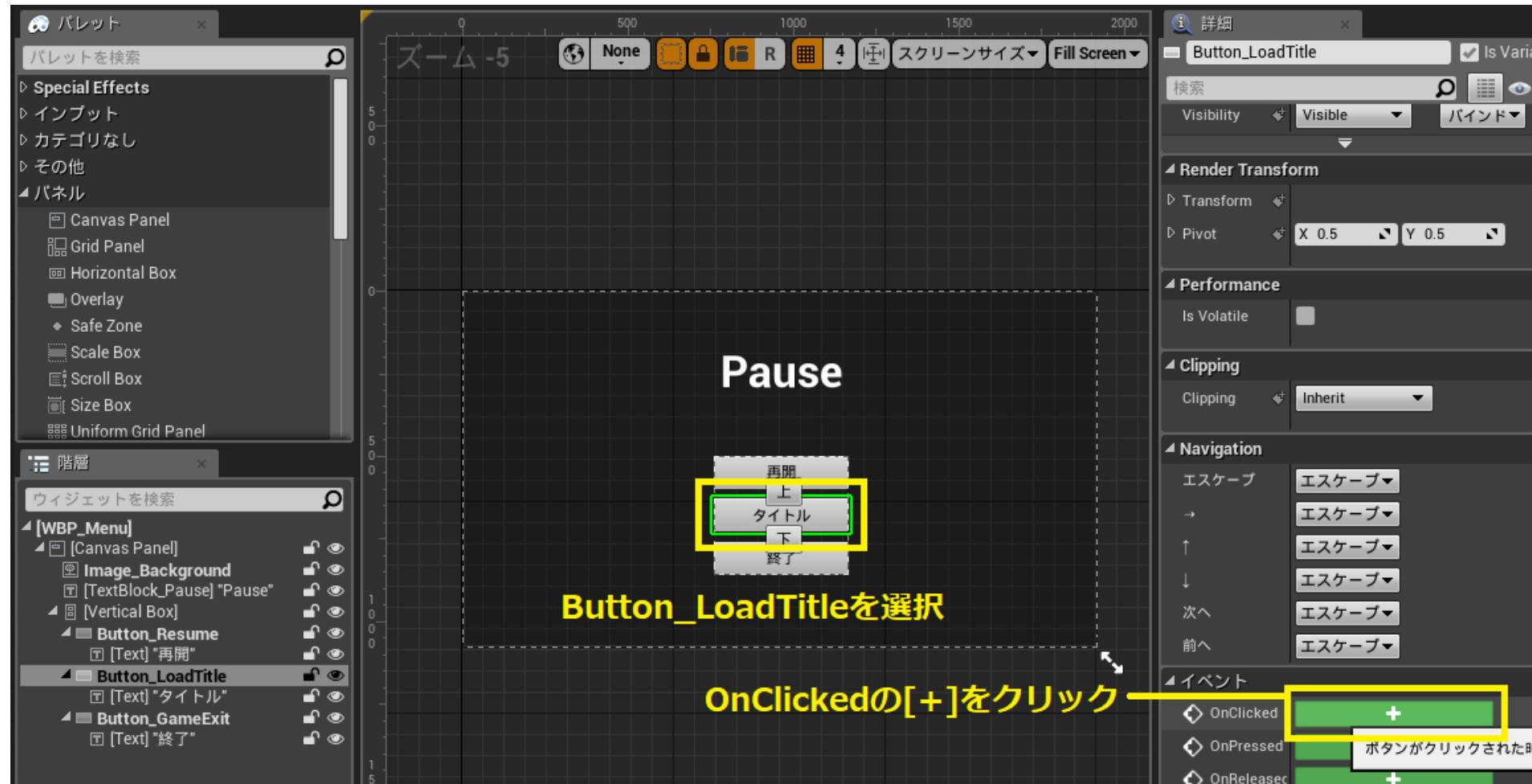
WBP_MenuのButton_ResumeにOnClickedイベントを追加する 1



WBP_MenuのButton_ResumeにOnClickedイベントを追加する 2



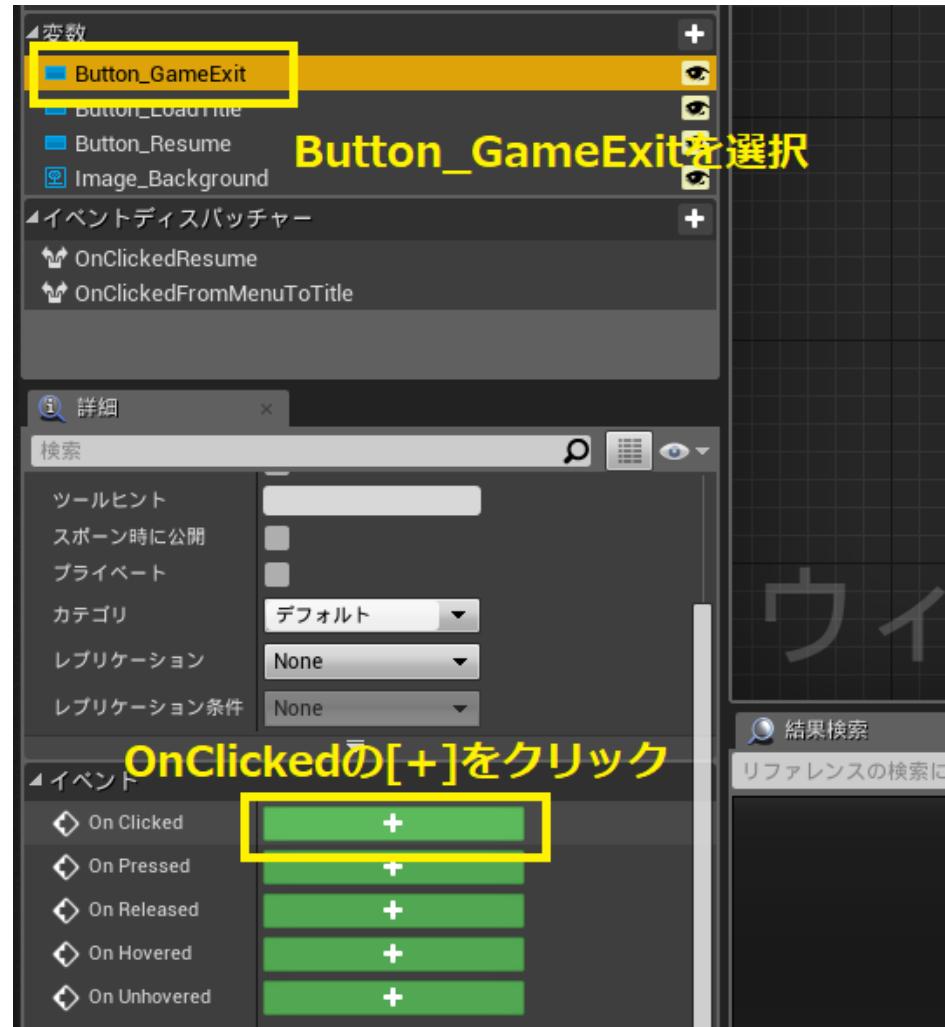
WBP_MenuのButton_LoadTitleにOnClickedイベントを追加する 1



WBP_MenuのButton_LoadTitleにOnClickedイベントを追加する 2



WBP_MenuのButton_GameExitにOnClickedイベントを追加する 1



デザイナーに移動しなくても、
変数のボタンを選択してOn Clickedイベントを
追加することが出来る

WBP_MenuのButton_GameExitにOnClickedイベントを追加する 2



コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

12. メニュー画面の作成

12.1 WBP_Menuの作成

12.2 WBP_Menuを表示

12.3 WBP_Menuのボタンからイベントディスパッチャを呼び出す

12.4 WBP_Menuのボタンイベントをバインドする

12.4.1 再開ボタンのイベントにバインド

12.4.2 タイトルボタンのイベントにバインド

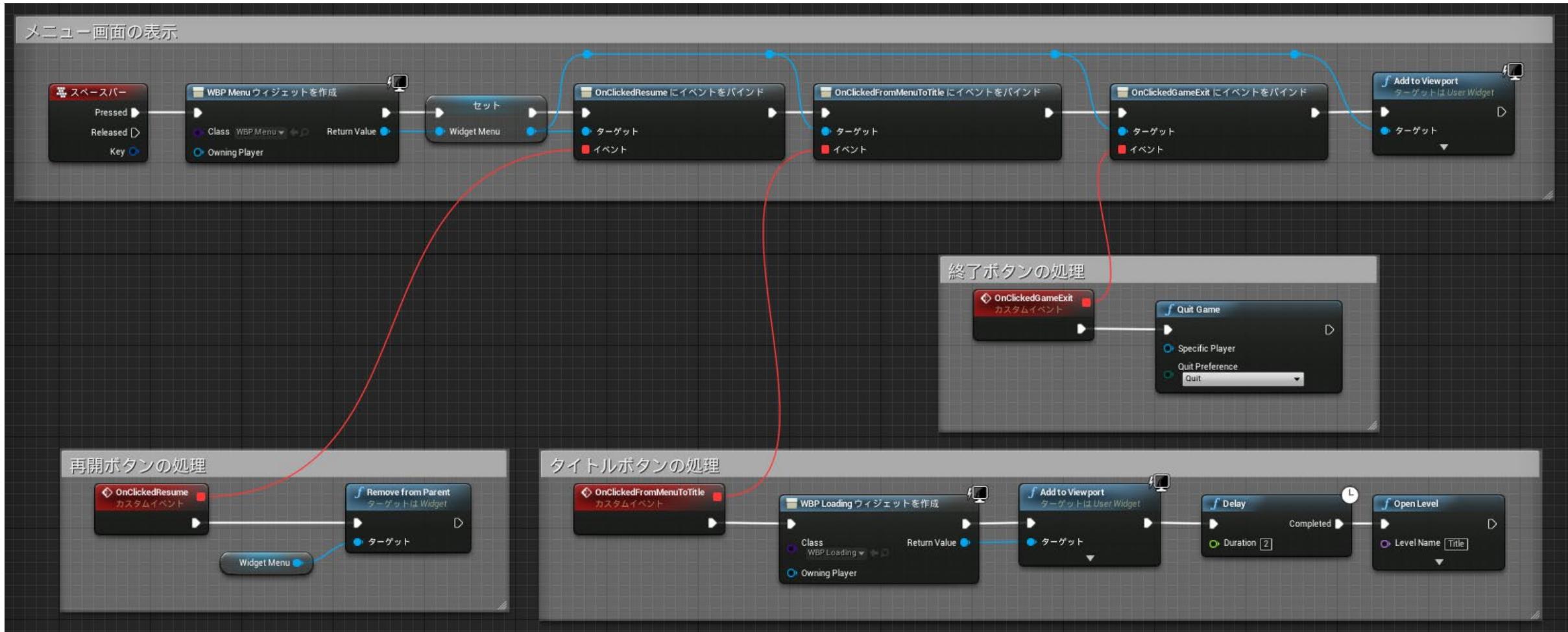
12.4.3 終了ボタンのイベントにバインド

メニュー画面の各ボタン動作



ボタン	動作
再開	メニュー画面を消す
タイトル	Titleレベルに遷移する
終了	ゲームを終了する

ボタンのイベントバインディング処理 完成図



12. メニュー画面の作成

12.1 WBP_Menuの作成

12.2 WBP_Menuを表示

12.3 WBP_Menuのボタンからイベントディスパッチャを呼び出す

12.4 WBP_Menuのボタンイベントをバインドする

12.4.1 再開ボタンのイベントにバインド

12.4.2 タイトルボタンのイベントにバインド

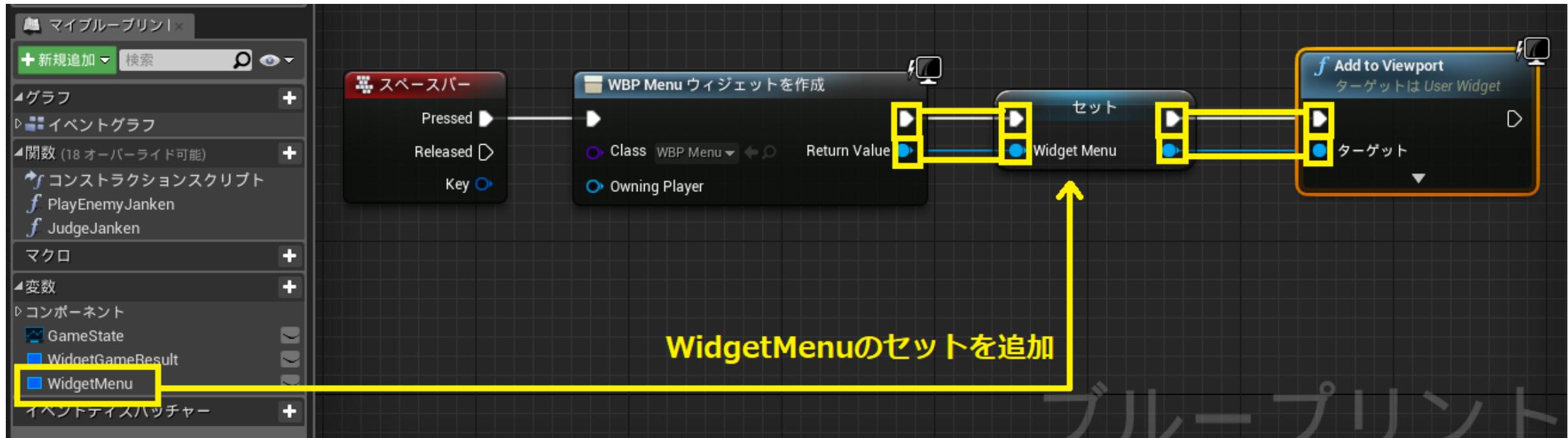
12.4.3 終了ボタンのイベントにバインド

BP_JankenGameControllerに変数を追加する

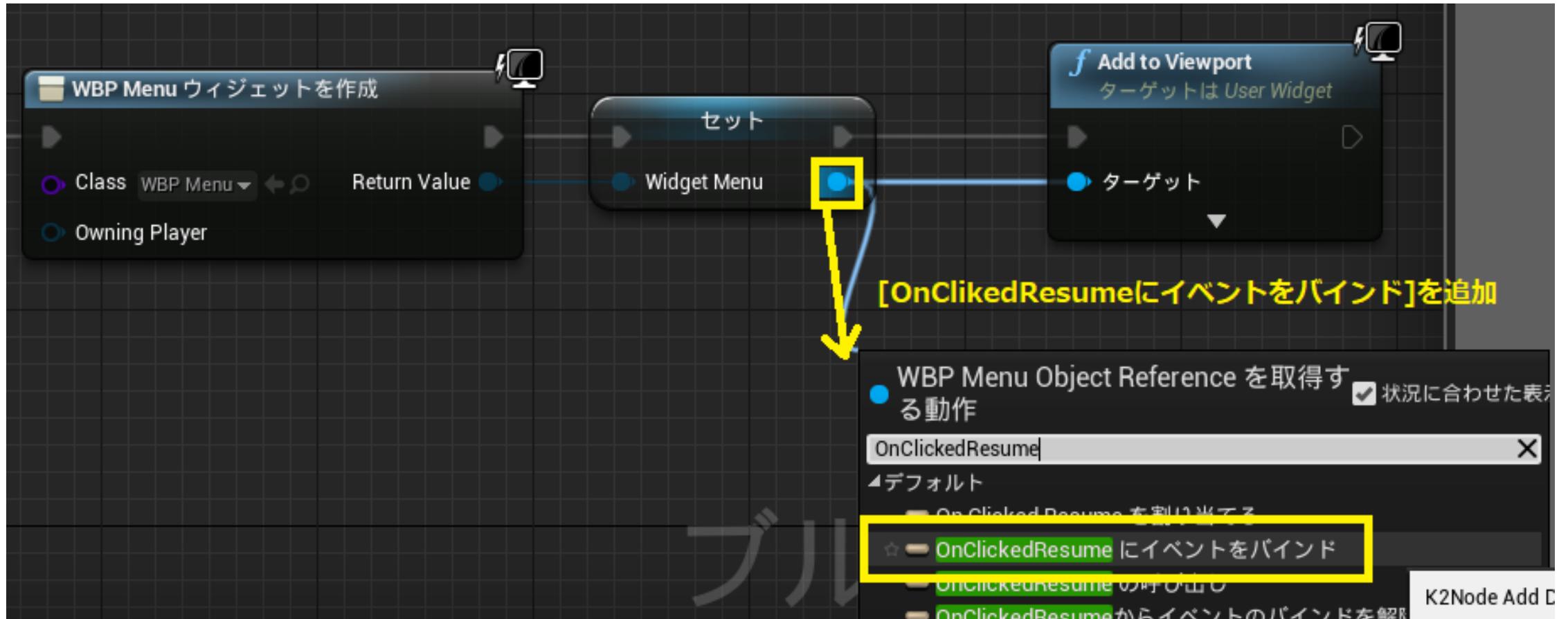


変数名	変数の型	デフォルト値
GameState	BP_GameState	-(設定なし)
WidgetGameResult	WBP_GameResult (Object Reference)	-(設定なし)
WidgetMenu	WBP_Menu (Object Reference)	-(設定なし)

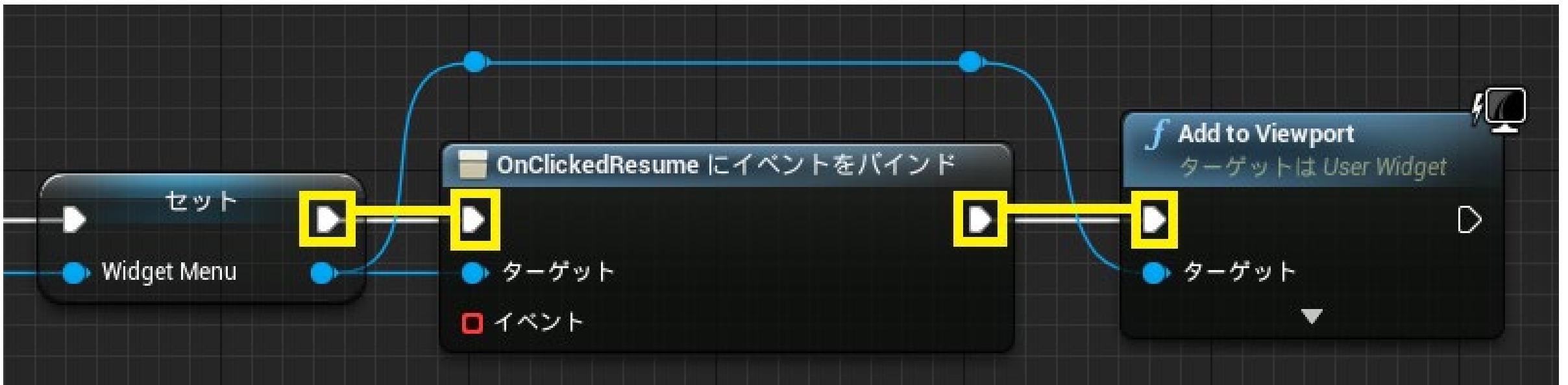
再開ボタンのClickイベント:OnClickedResumeにバインドする 1



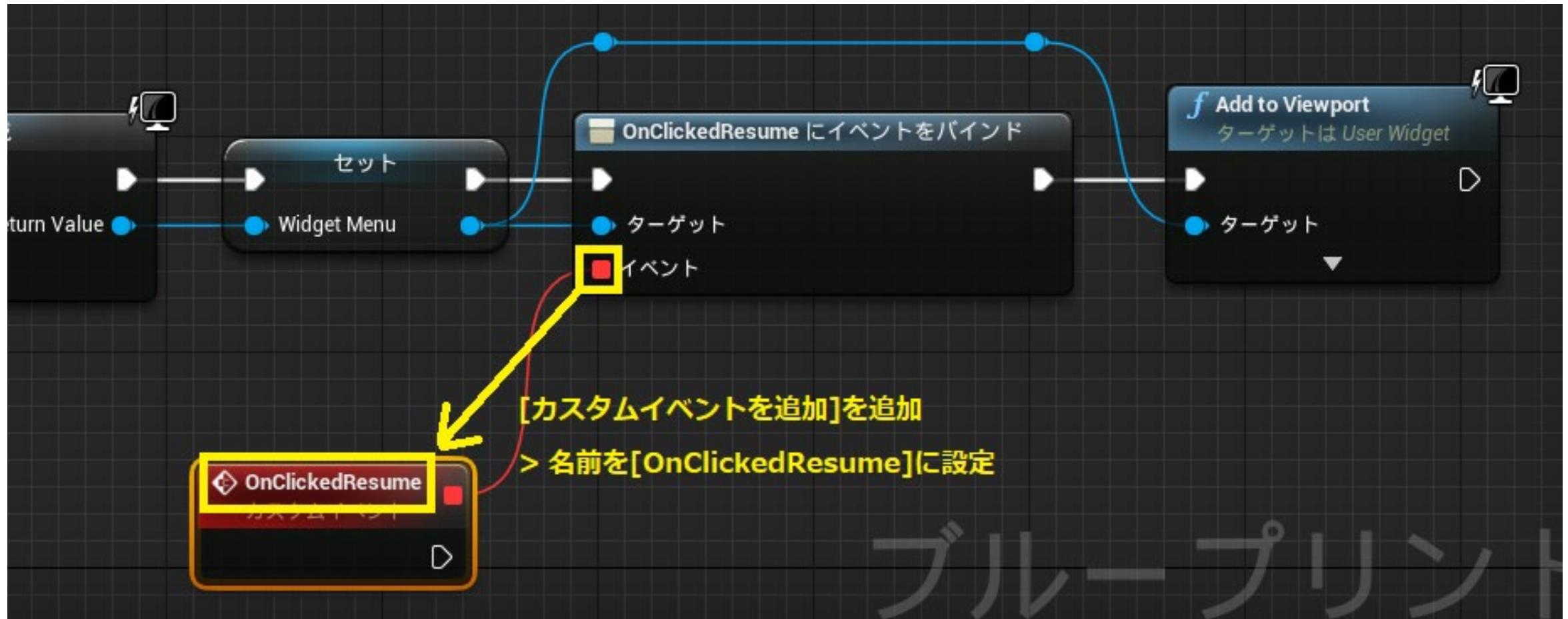
再開ボタンのClickイベント:OnClickedResumeにバインドする 2



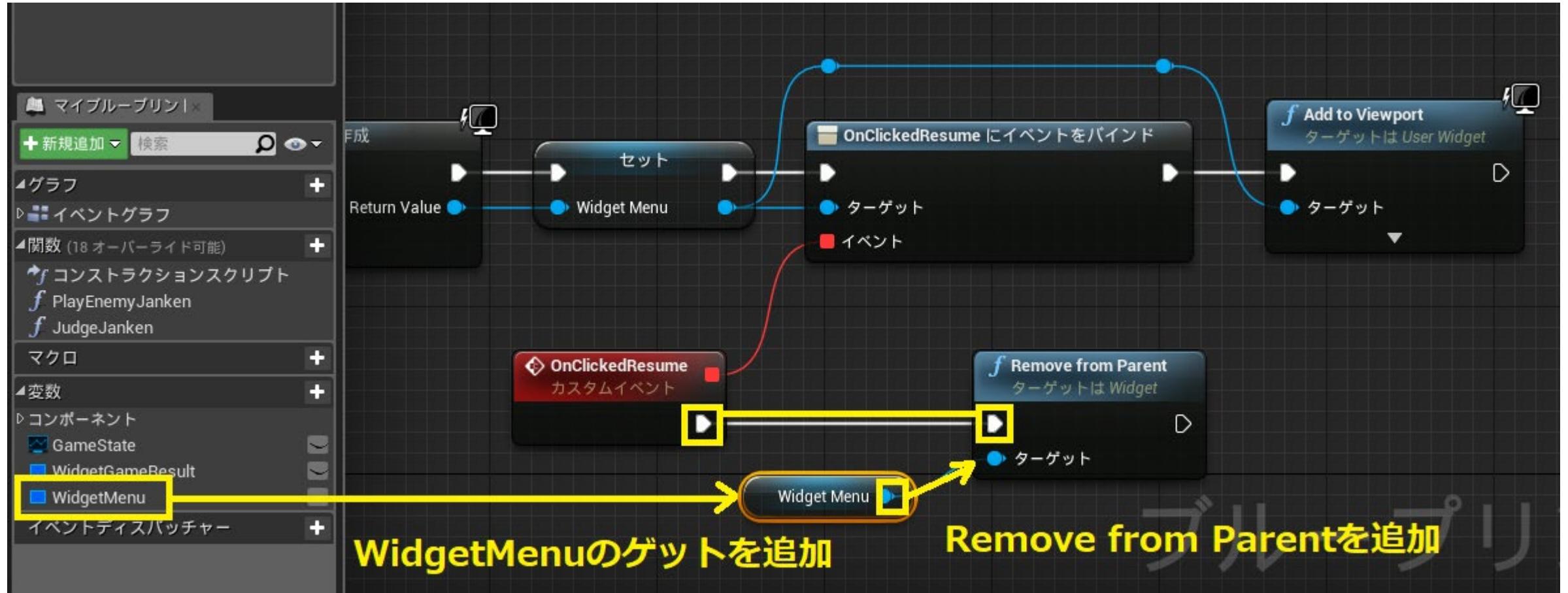
再開ボタンのClickイベント:OnClickedResumeにバインドする 3



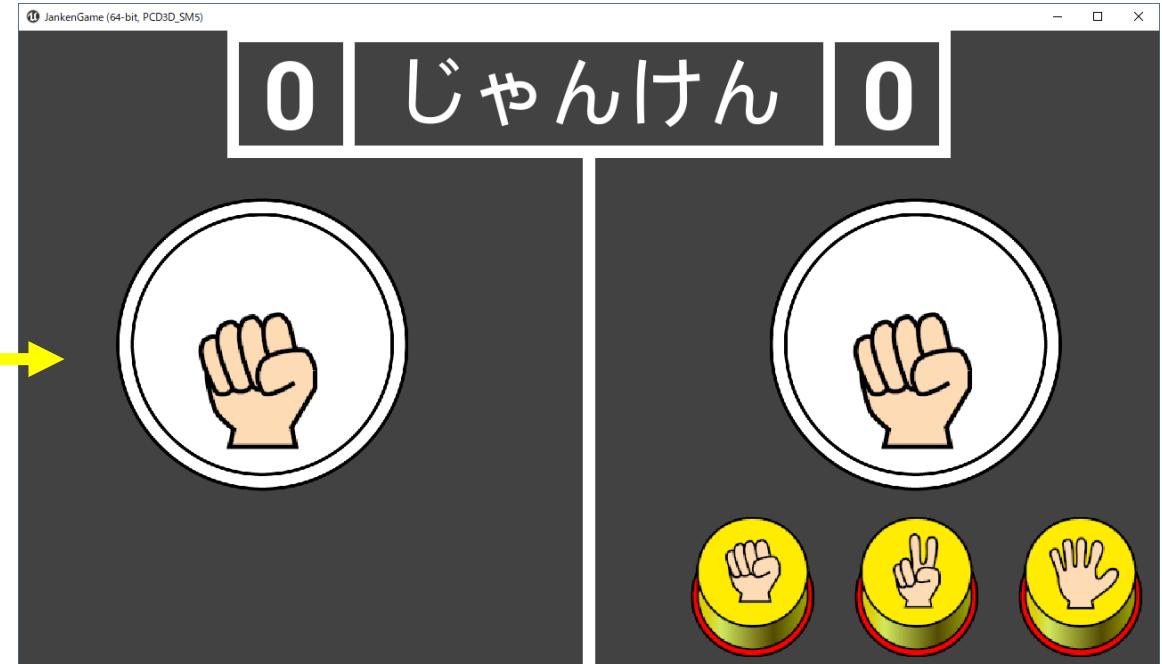
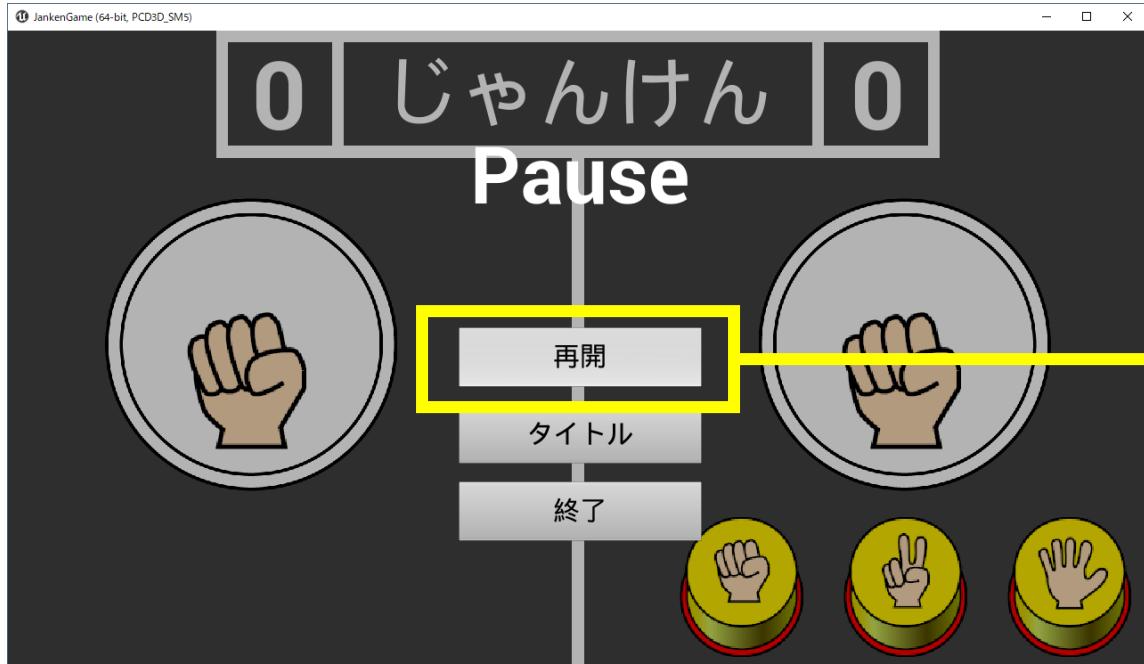
再開ボタンのClickイベント:OnClickedResumeにバインドする 4



再開ボタンのClickイベント:OnClickedResumeにバインドする 5

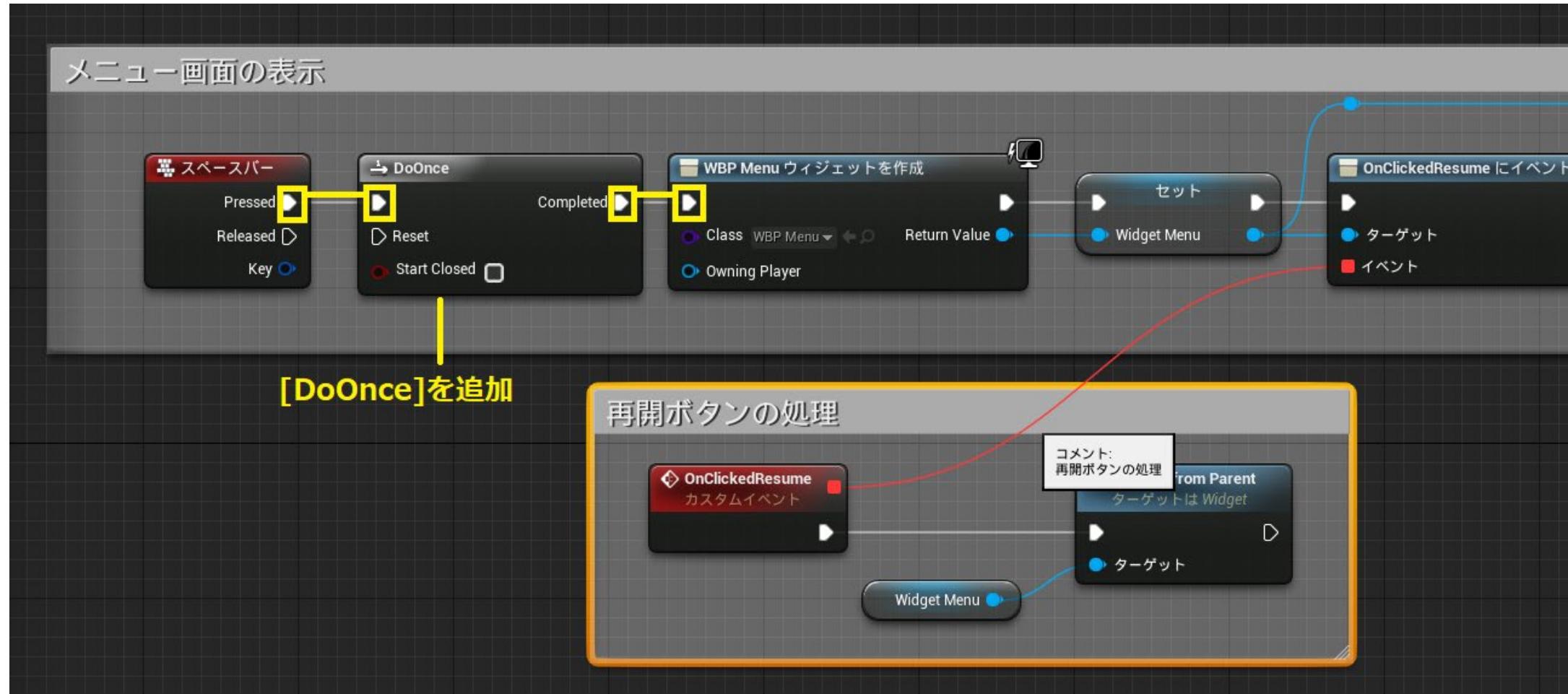


プレイして確認する

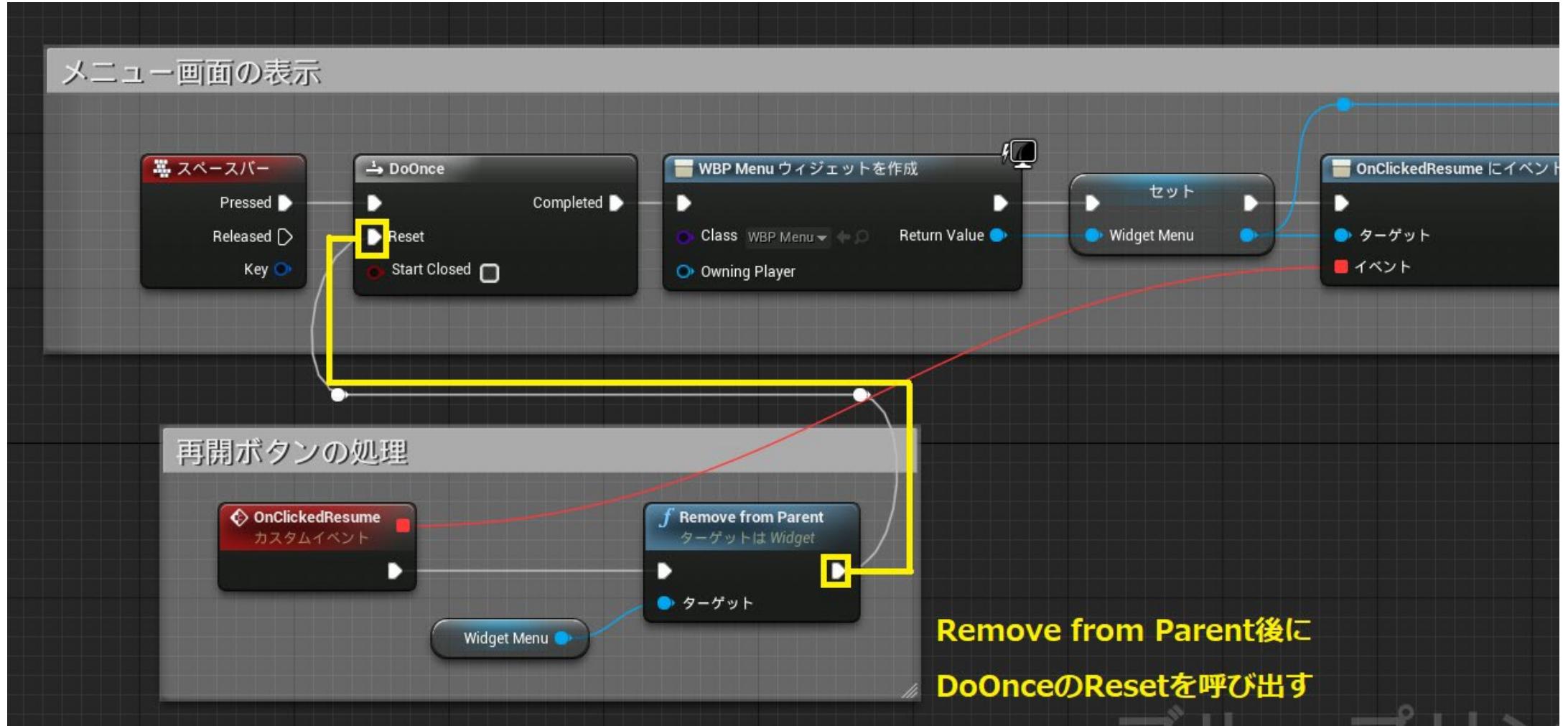


メニューを表示して再開ボタンをクリックすると
メニューが消えてゲームを再開することが出来る

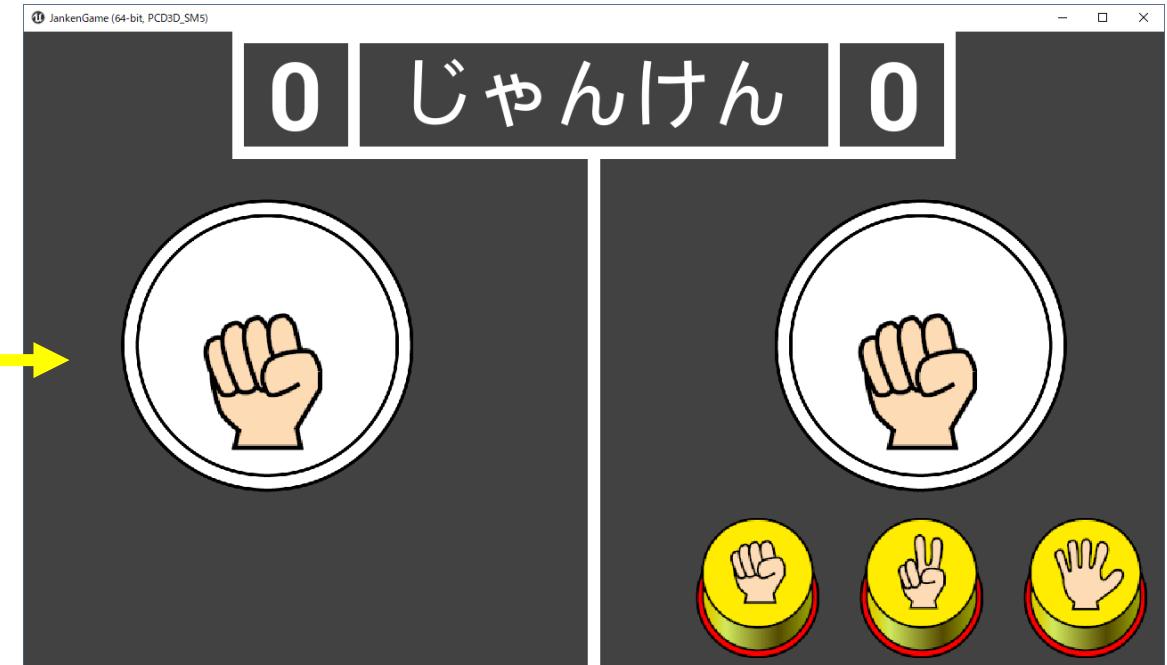
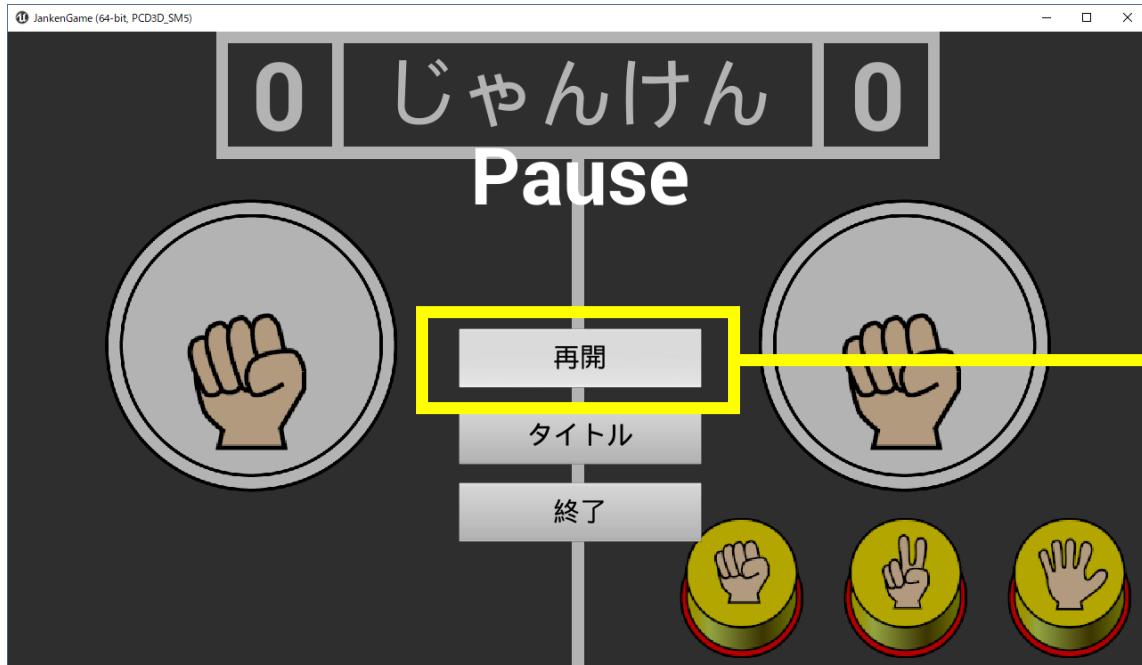
スペースバーを複数押してもメニューが一度だけ表示されるようにする 1



スペースバーを複数押してもメニューが一度だけ表示されるようにする 2



プレイして確認する



スペースバーを連打しても再開をクリックするまで、メニューを再び表示することが出来なくなる

12. メニュー画面の作成

12.1 WBP_Menuの作成

12.2 WBP_Menuを表示

12.3 WBP_Menuのボタンからイベントディスパッチャを呼び出す

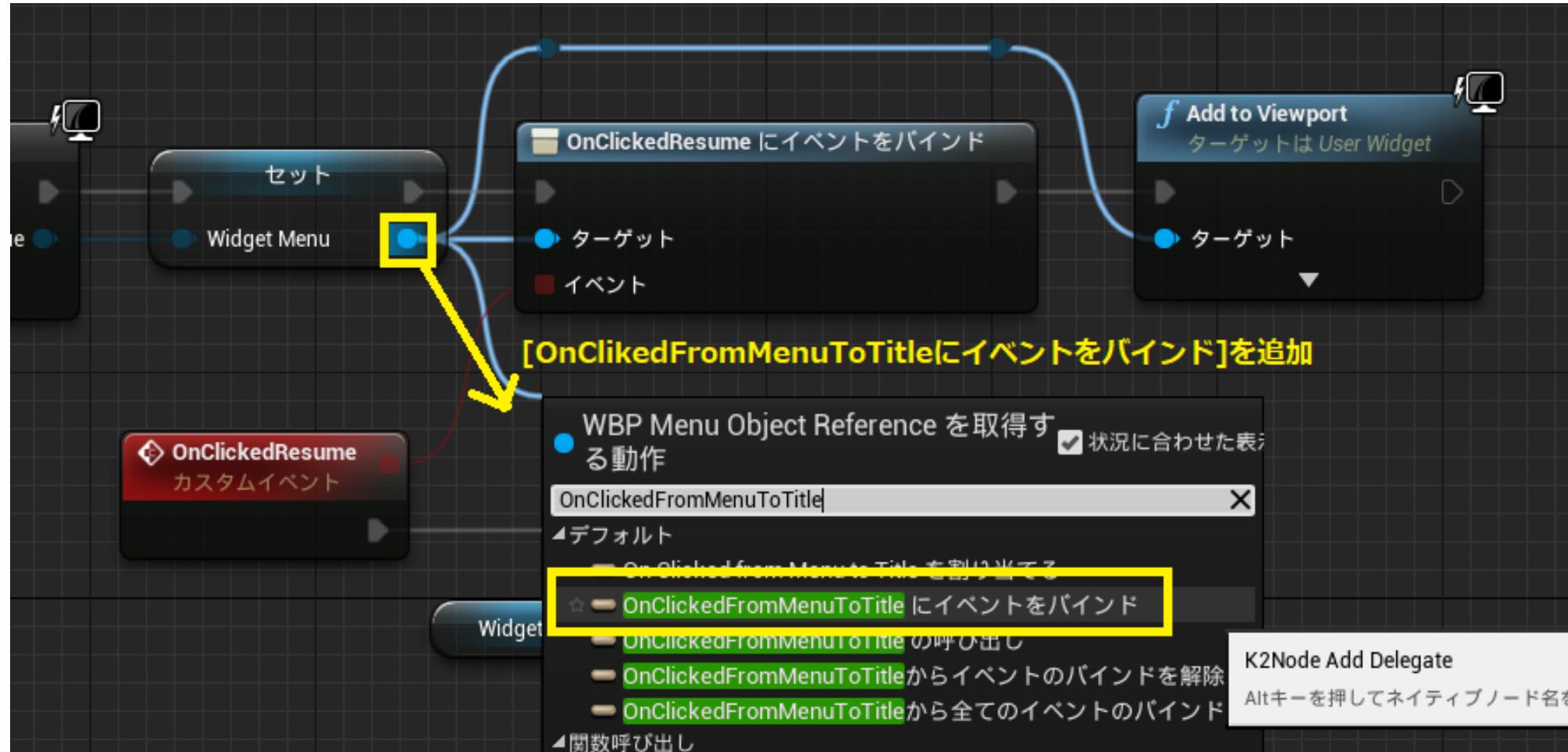
12.4 WBP_Menuのボタンイベントをバインドする

 12.4.1 再開ボタンのイベントにバインド

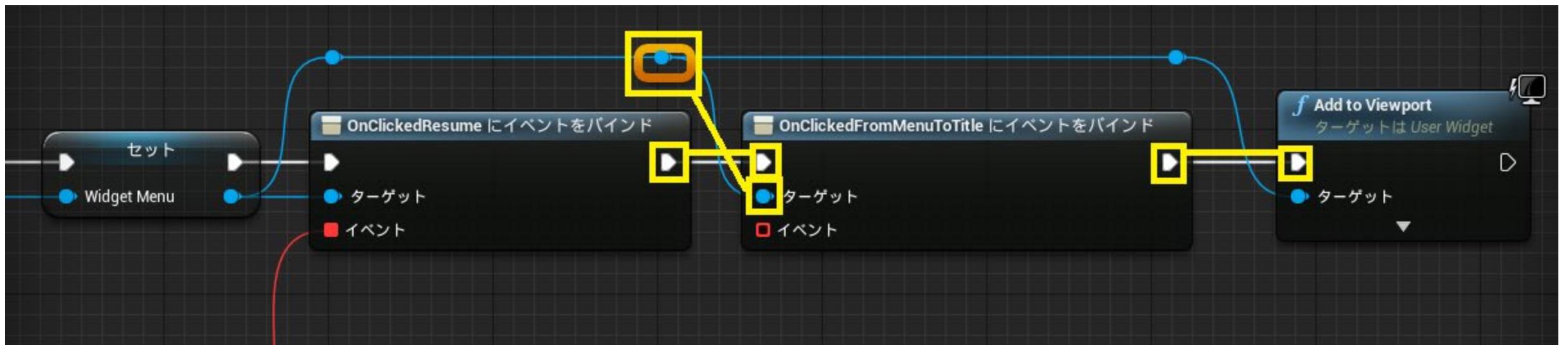
 12.4.2 タイトルボタンのイベントにバインド

 12.4.3 終了ボタンのイベントにバインド

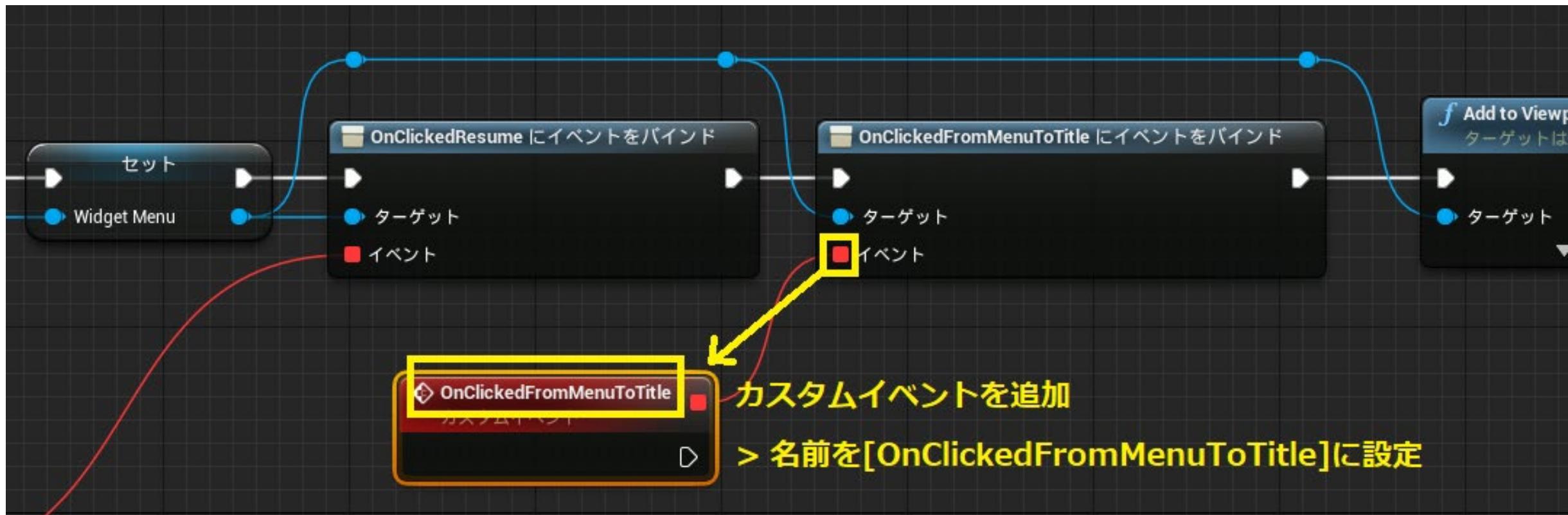
タイトルボタンのClickイベント:OnClickedFromMenuToTitleにバインドする 1



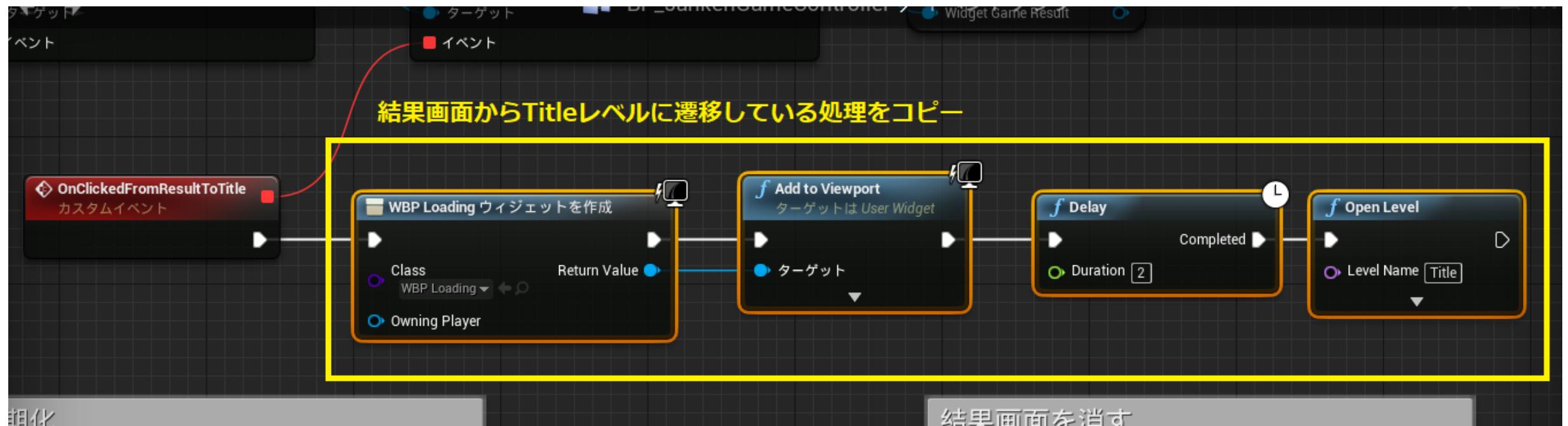
タイトルボタンのClickイベント:OnClickedFromMenuToTitleにバインドする 2



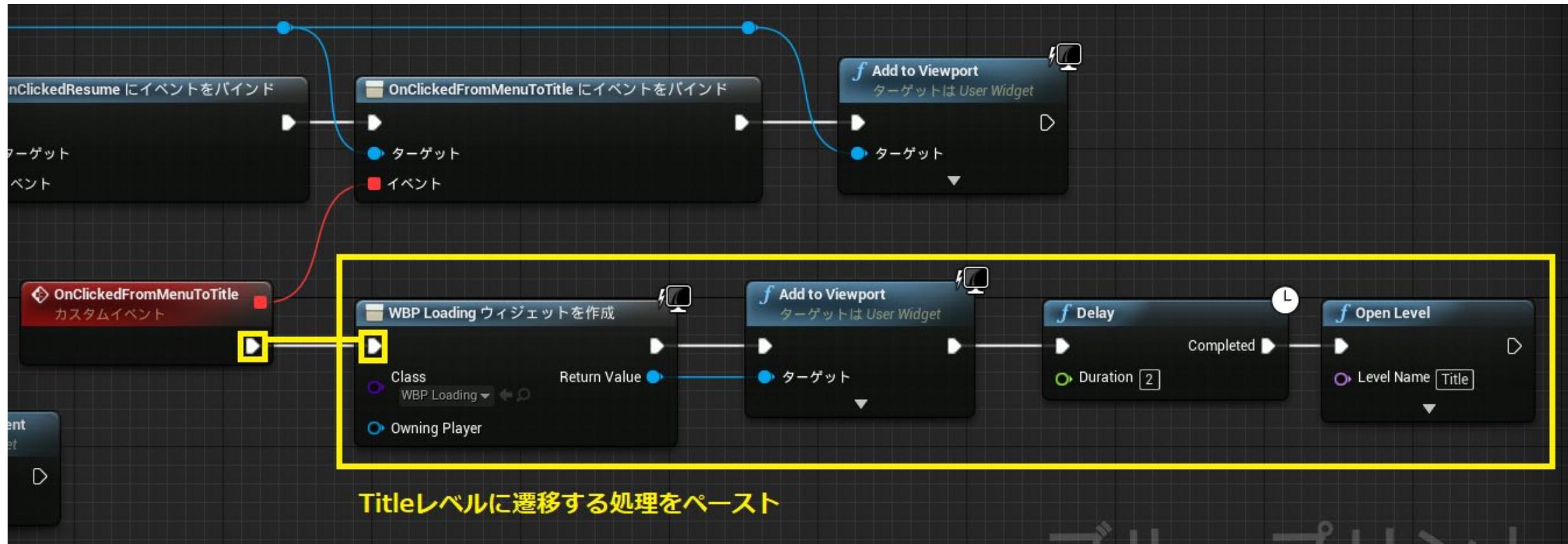
タイトルボタンのClickイベント:OnClickedFromMenuItemにバインドする 3



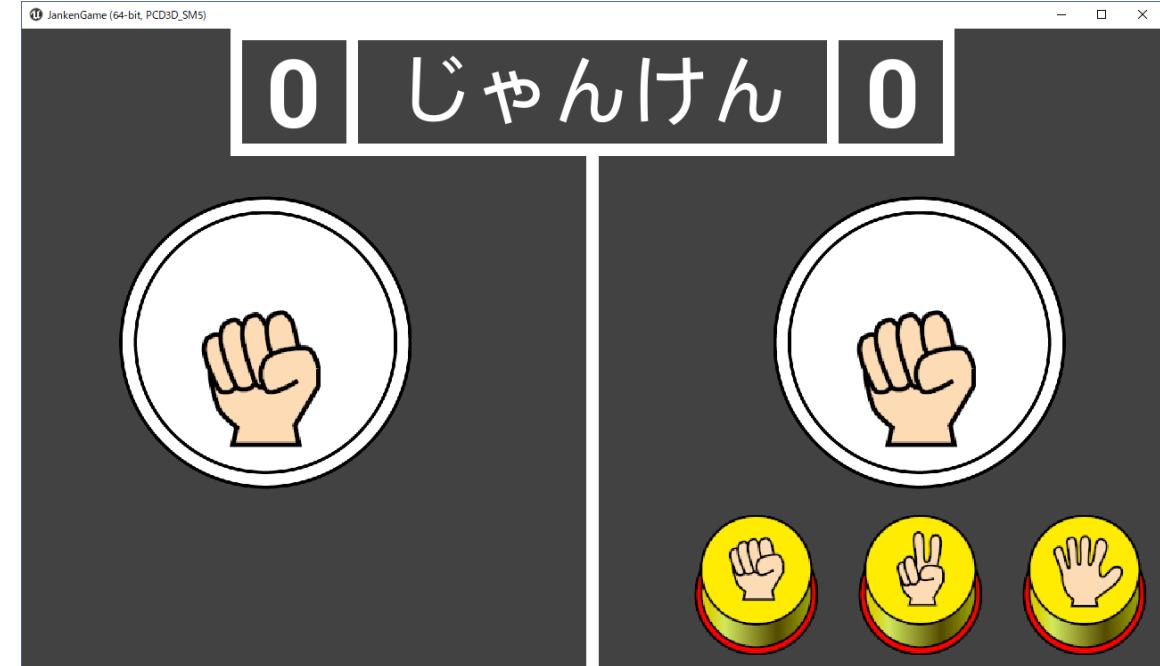
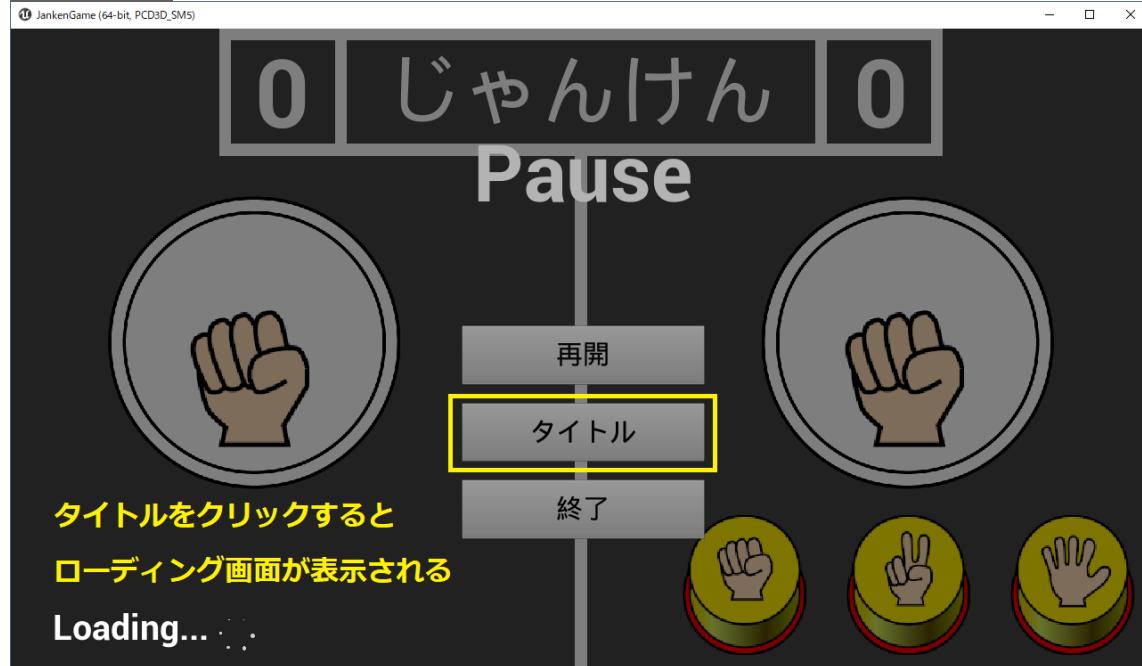
タイトルボタンのClickイベント:OnClickedFromMenuToTitleにバインドする 4



タイトルボタンのClickイベント:OnClickedFromMenuToTitleにバインドする 5



プレイして確認する



メニューを表示してタイトルボタンをクリックすると
ローディング画面が表示されてTitleレベルに遷移する

12. メニュー画面の作成

12.1 WBP_Menuの作成

12.2 WBP_Menuを表示

12.3 WBP_Menuのボタンからイベントディスパッチャを呼び出す

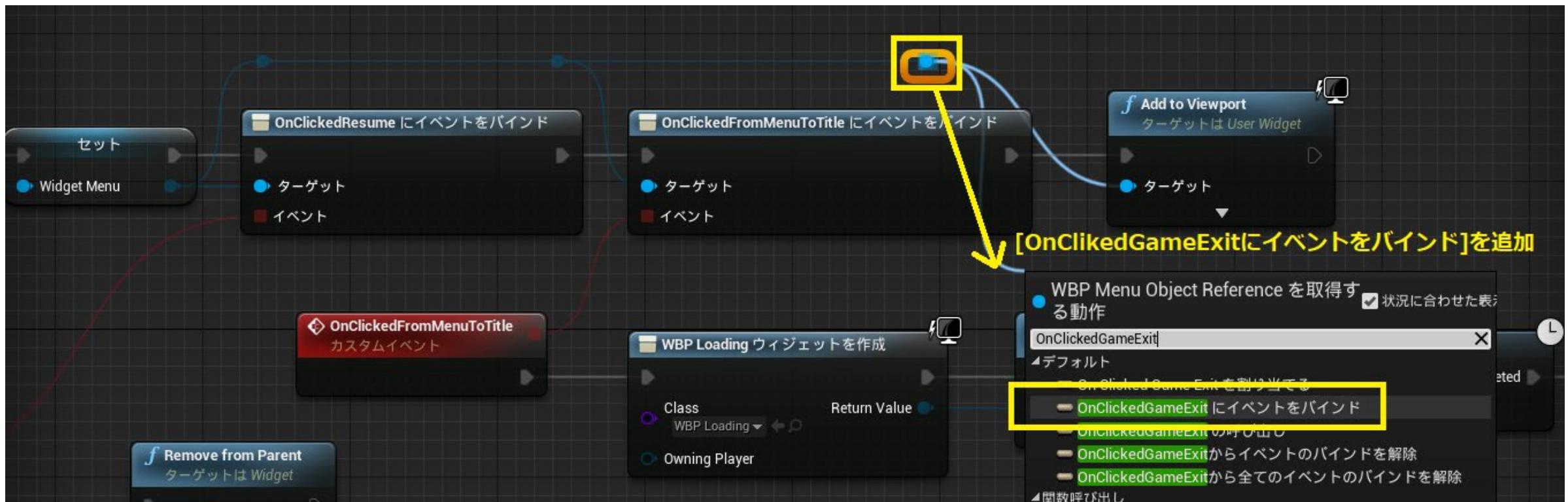
12.4 WBP_Menuのボタンイベントをバインドする

 12.4.1 再開ボタンのイベントにバインド

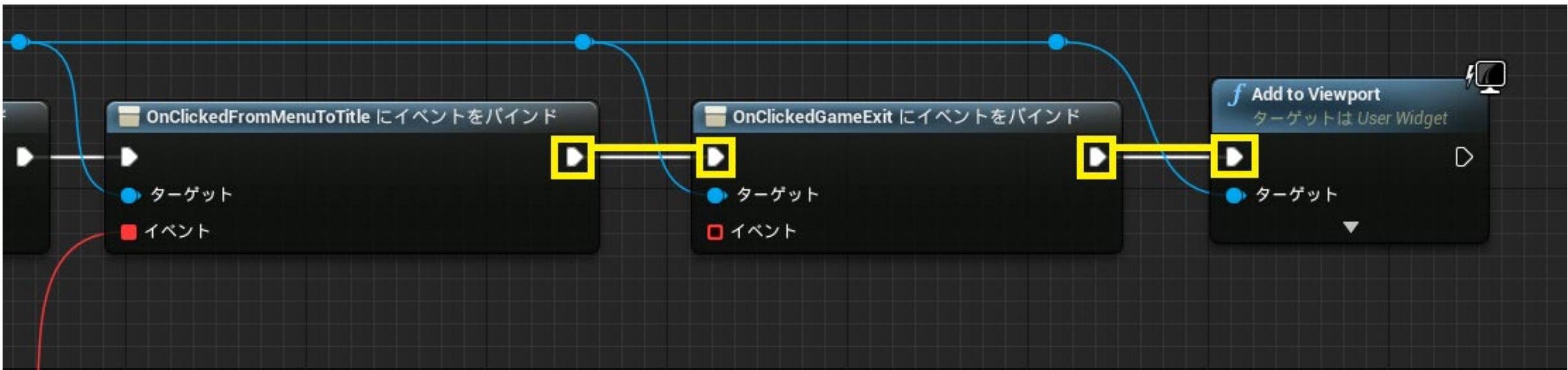
 12.4.2 タイトルボタンのイベントにバインド

 12.4.3 終了ボタンのイベントにバインド

終了ボタンのClickイベント:OnClickedGameExitにバインドする 1



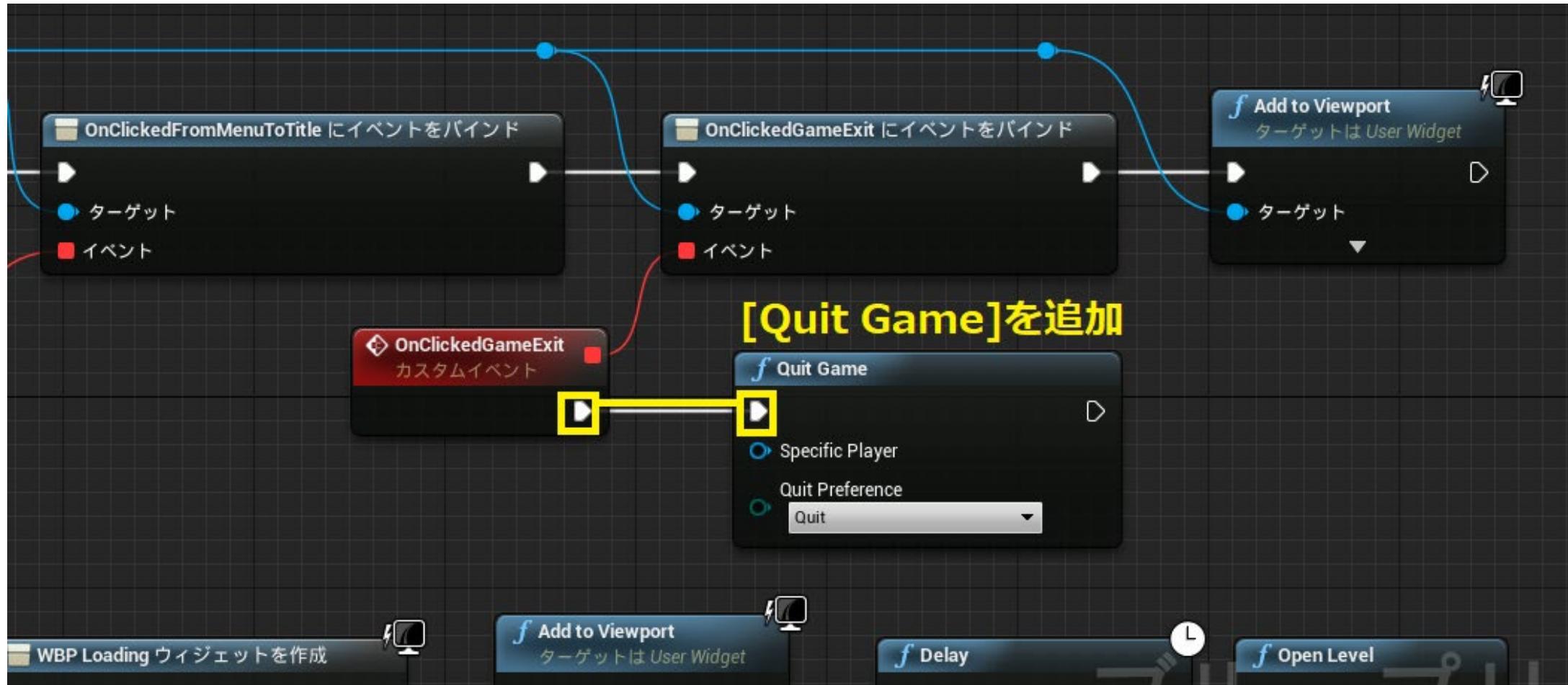
終了ボタンのClickイベント:OnClickedGameExitにバインドする 2



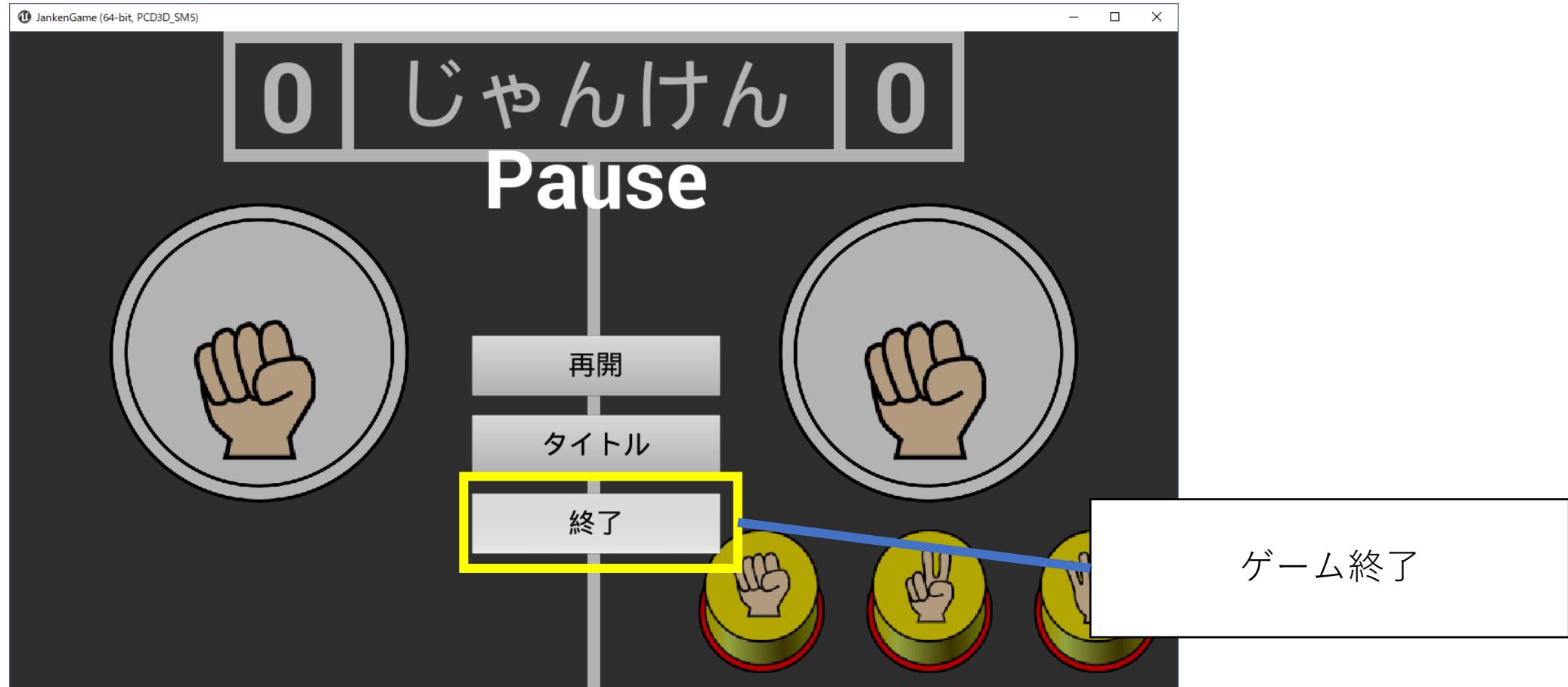
終了ボタンのClickイベント:OnClickedGameExitにバインドする 3



終了ボタンのClickイベント:OnClickedGameExitにバインドする 4



プレイして確認する

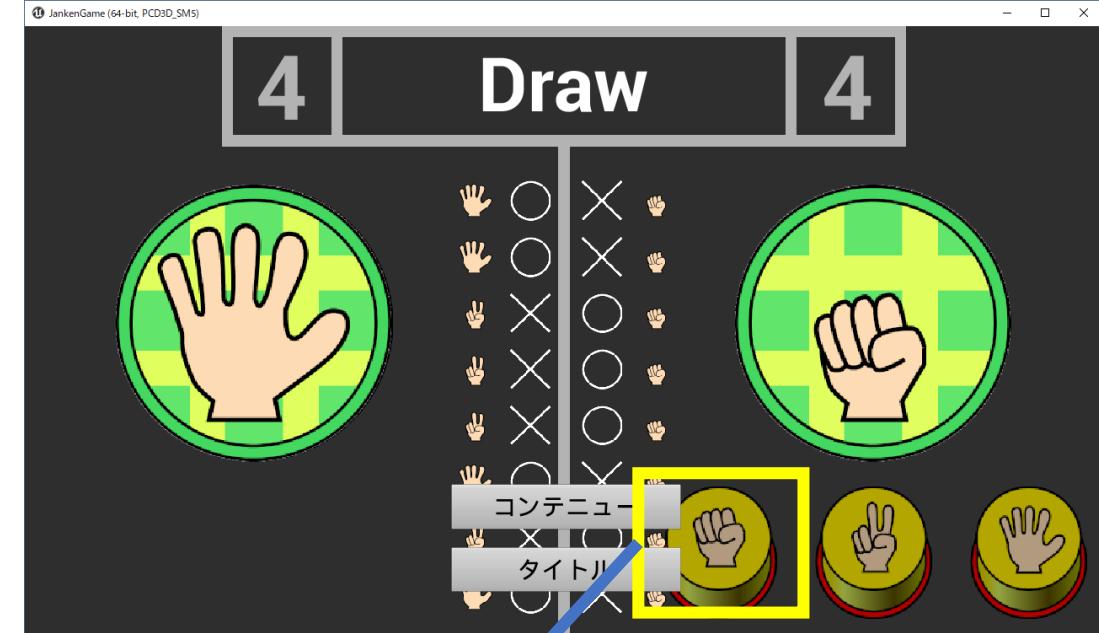


コンパイル > 保存



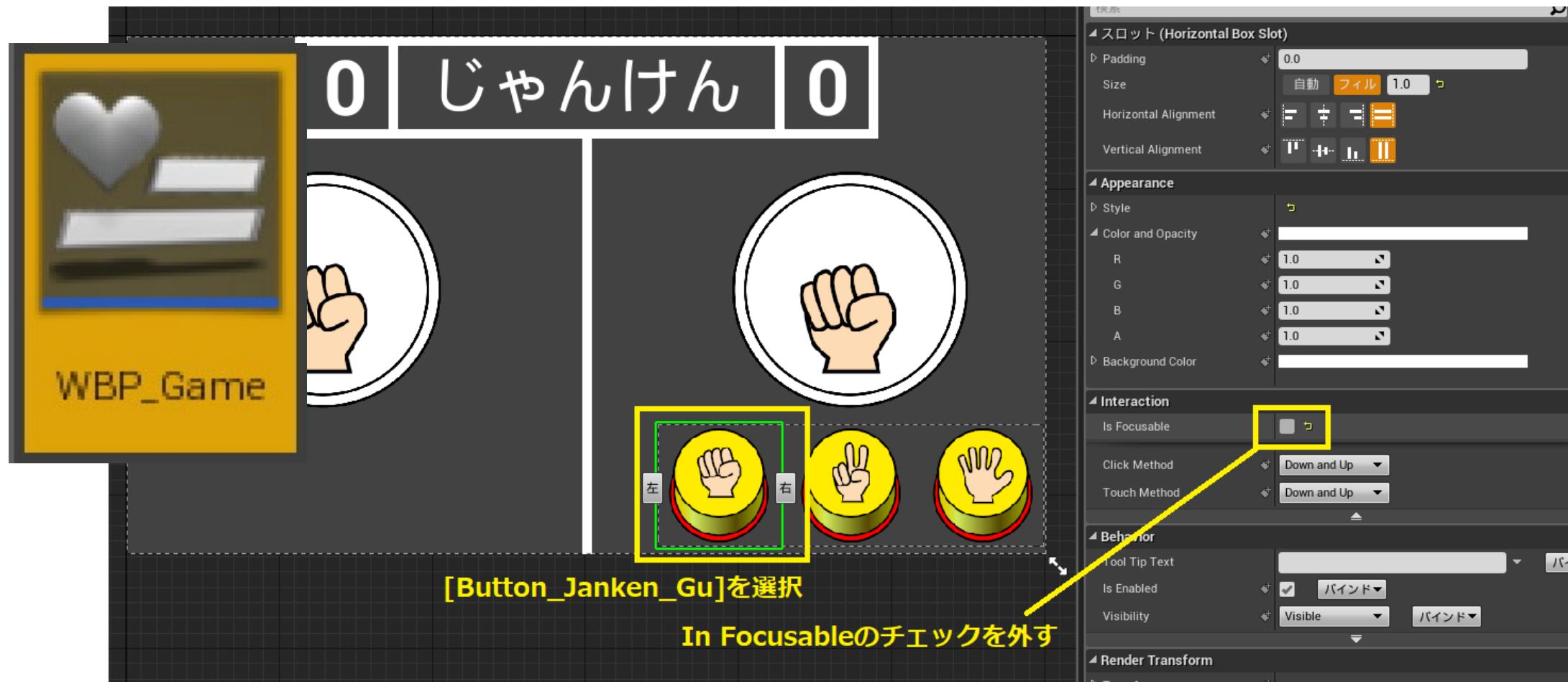
この後に別のブループリントを編集するため、コンパイル > 保存

メニュー表示時にボタンがスペースバーで押せてしまう

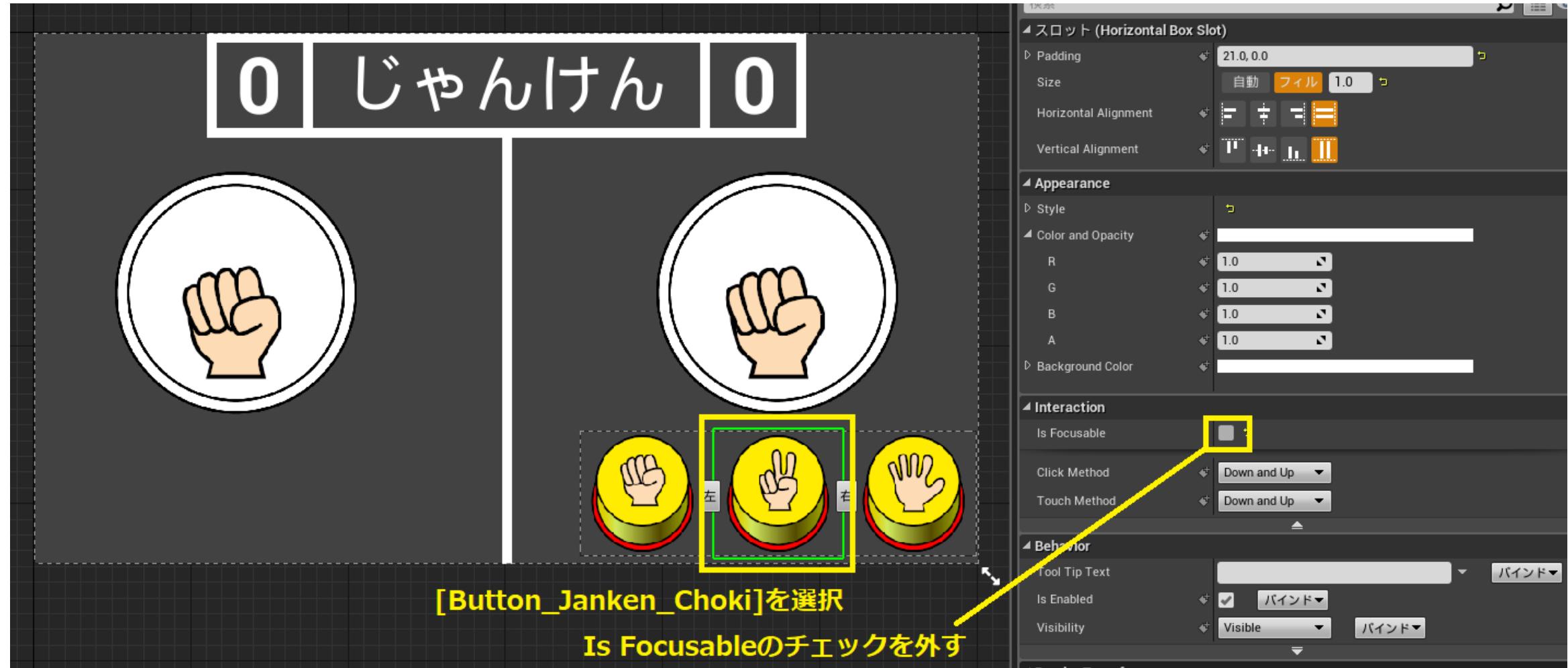


マウスカーソルを当てた状態でスペースボタンをプッシュするとボタンが押せてしまう

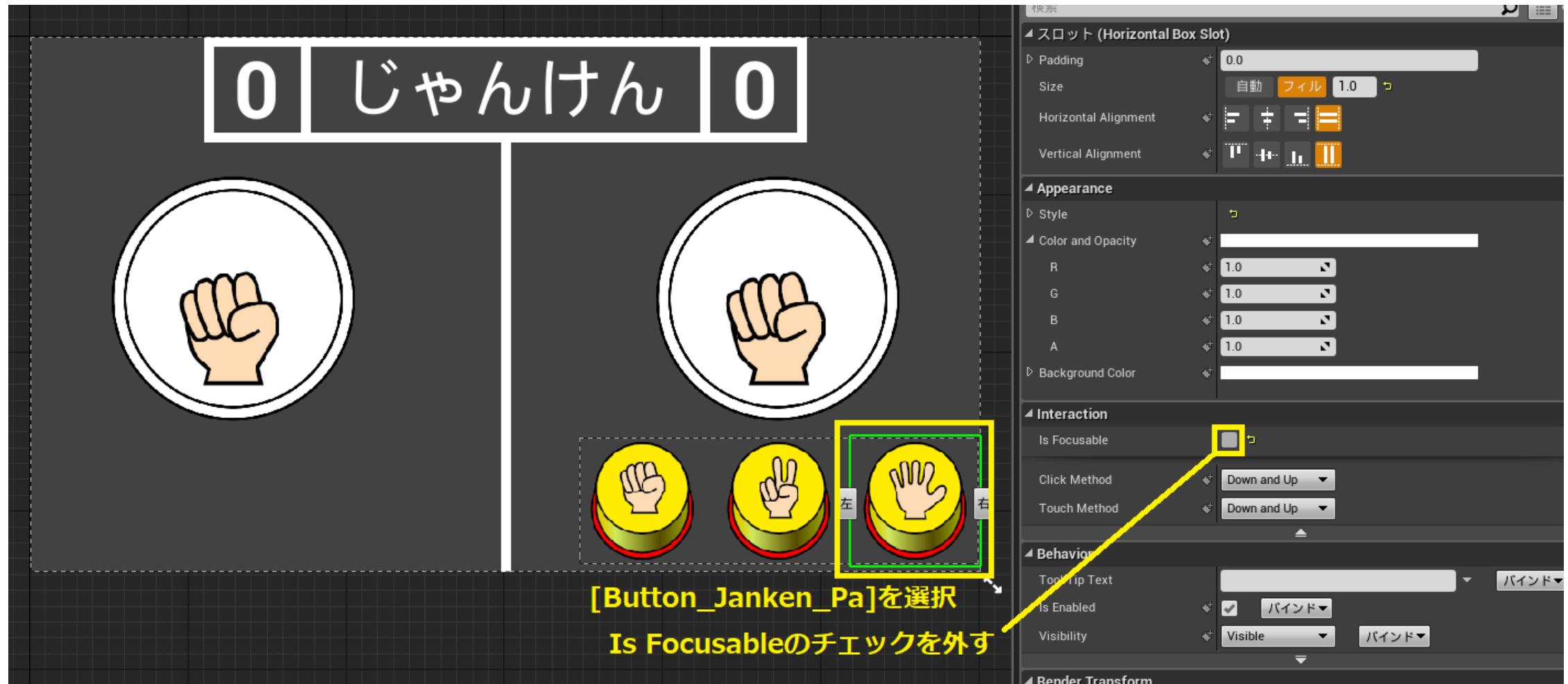
ボタンのIn Focusableのチェックを外す 1



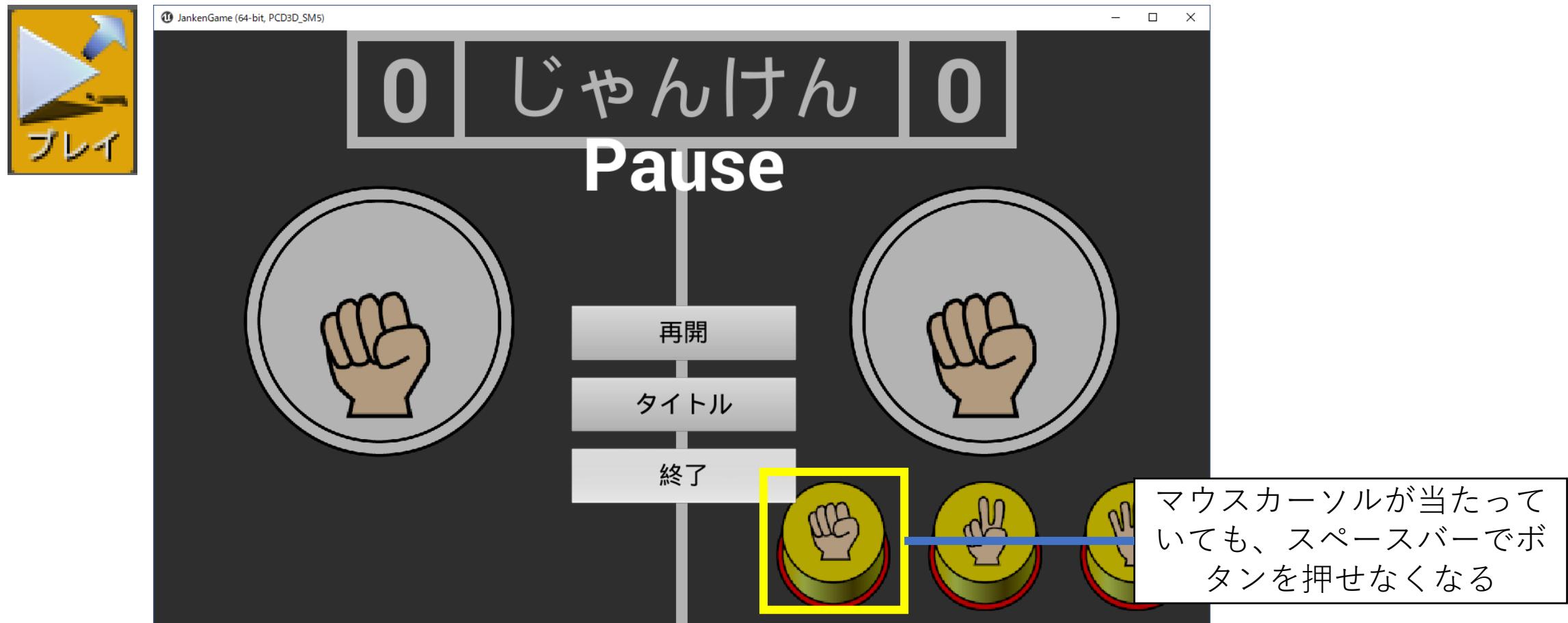
ボタンのIn Focusableのチェックを外す 2



ボタンのIn Focusableのチェックを外す 3



プレイして確認する



13. 音の追加

13. 音の追加

13.1 音データのインポート

13.2 BGMがループするようにキューを作成する

13.3 BGMを再生する

13.4 SEを再生する

13. 音の追加

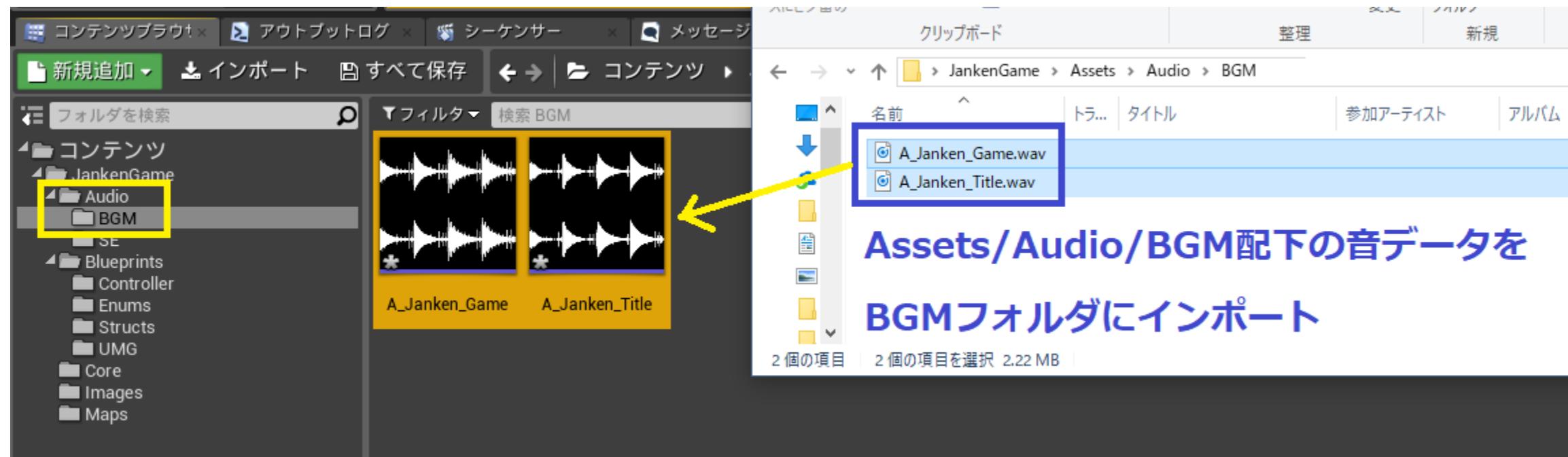
13.1 音データのインポート

13.2 BGMがループするようにキューを作成する

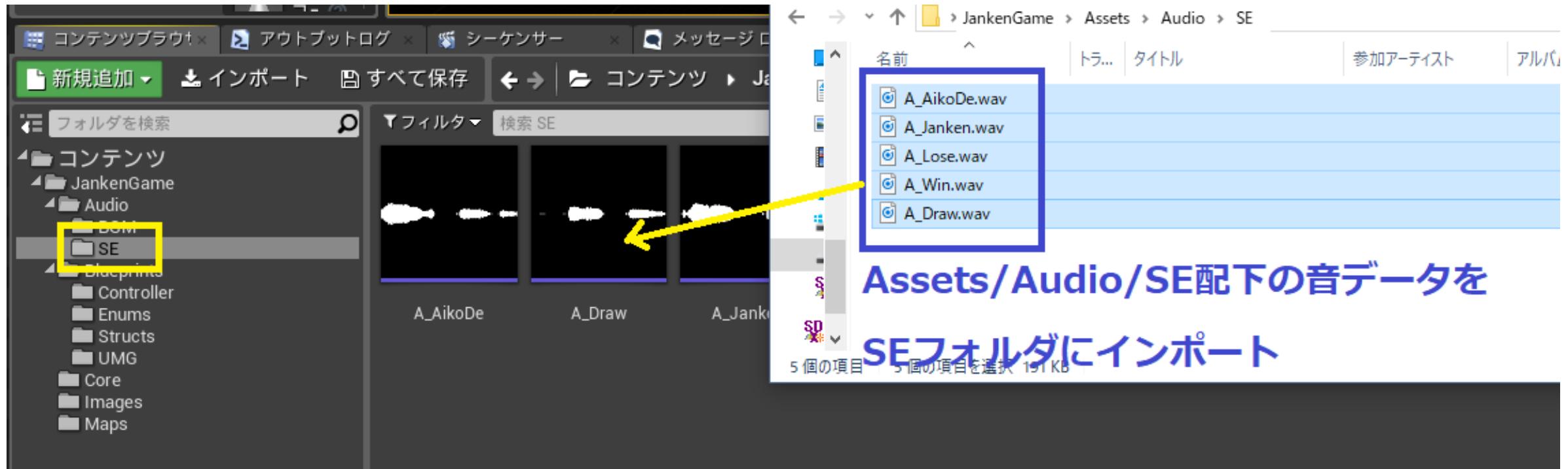
13.3 BGMを再生する

13.4 SEを再生する

音：BGMデータをインポートする



音：SEデータをインポートする



13. 音の追加

13.1 音データのインポート

13.2 BGMがループするようにキューを作成する

13.3 BGMを再生する

13.4 SEを再生する

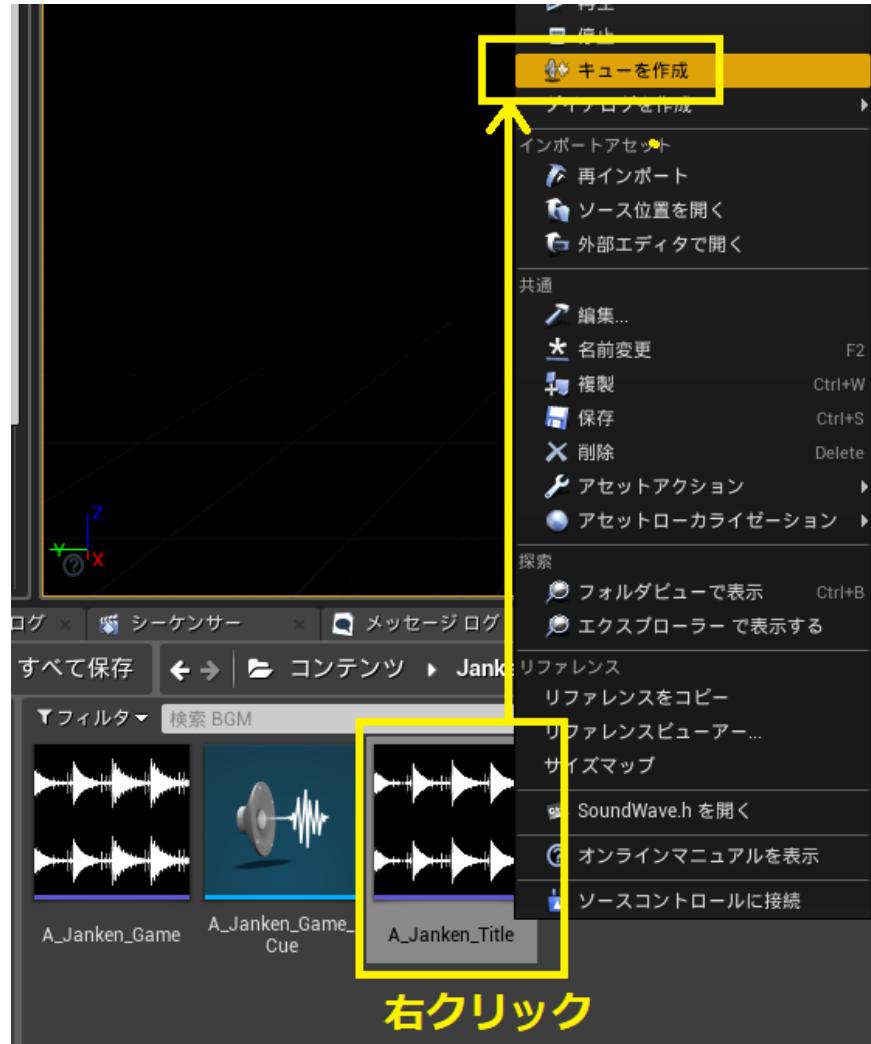
ループするBGM : A_Janken_Game_Cueを作成する 1



ループするBGM : A_Janken_Game_Cueを作成する 2



ループするBGM : A_Janken_Title_Cueを作成する 1



ループするBGM : A_Janken_Title_Cueを作成する 2



13. 音の追加

13.1 音データのインポート

13.2 BGMがループするようにキューを作成する

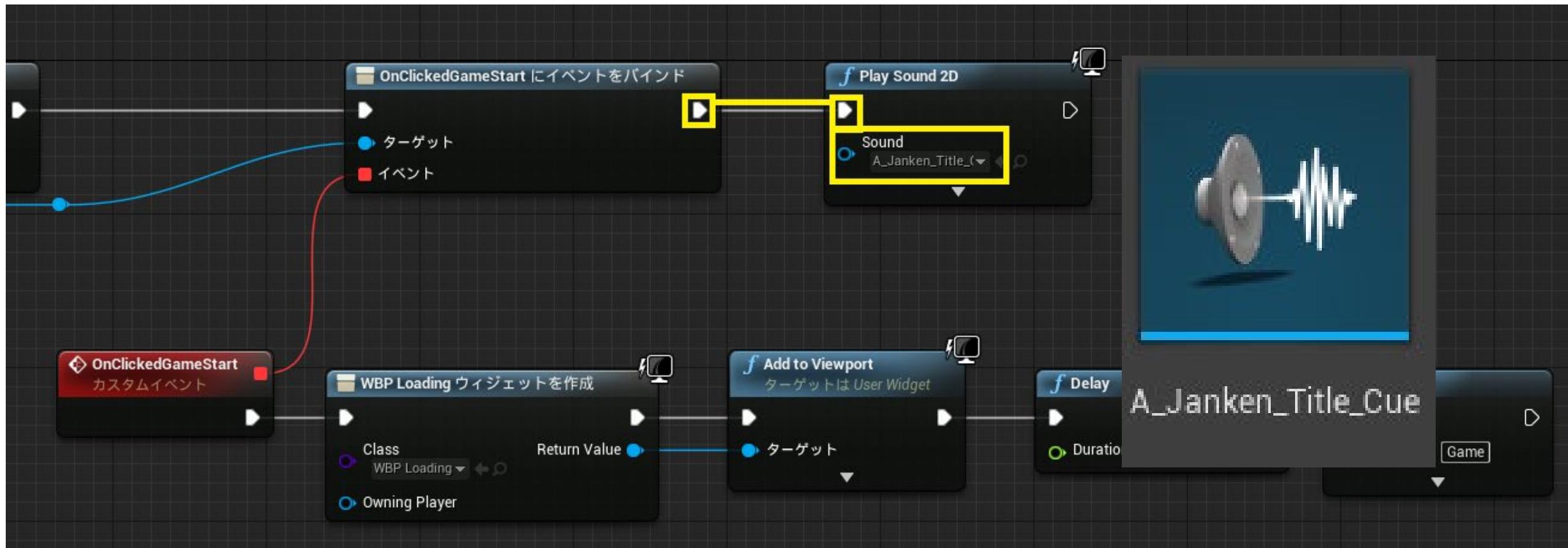
13.3 BGMを再生する

13.4 SEを再生する

BGM_Janken_Title_Cueを再生する 1



BGM_Janken_Title_Cueを再生する 2

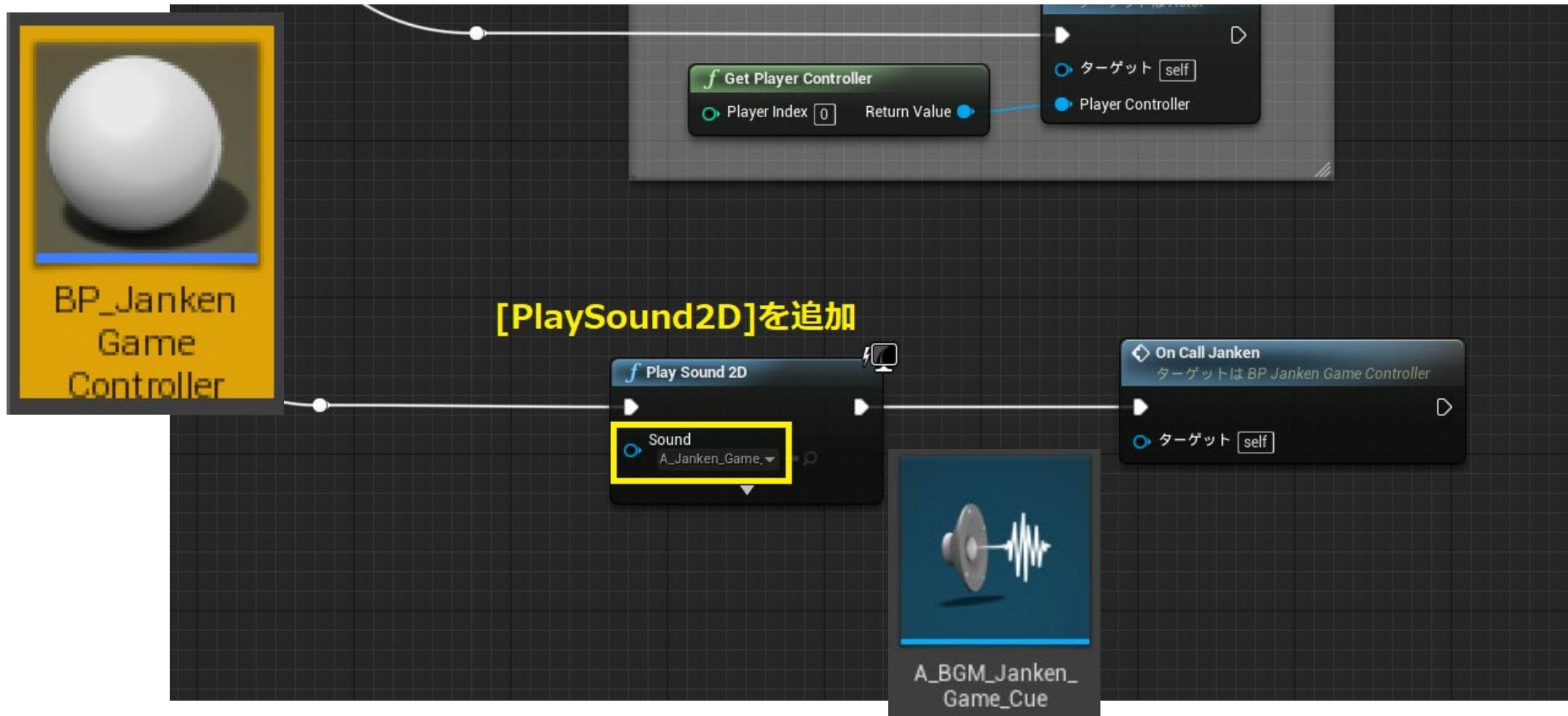


コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

BGM_Janken_Game_Cueを再生する 1

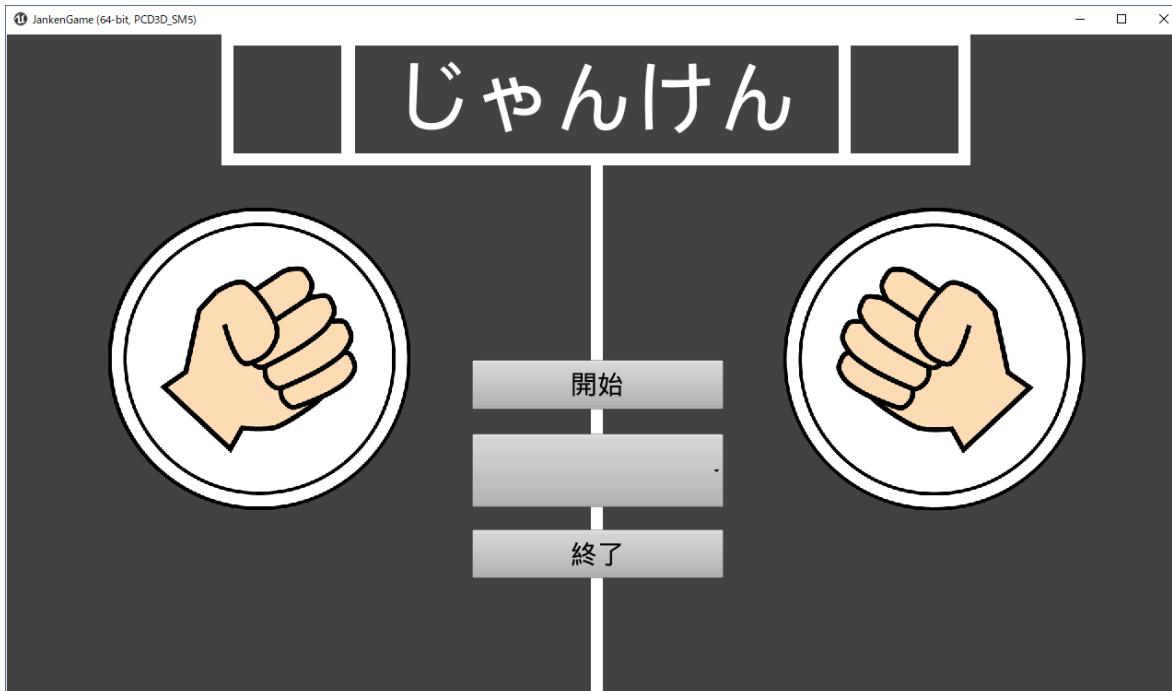


コンパイル > 保存

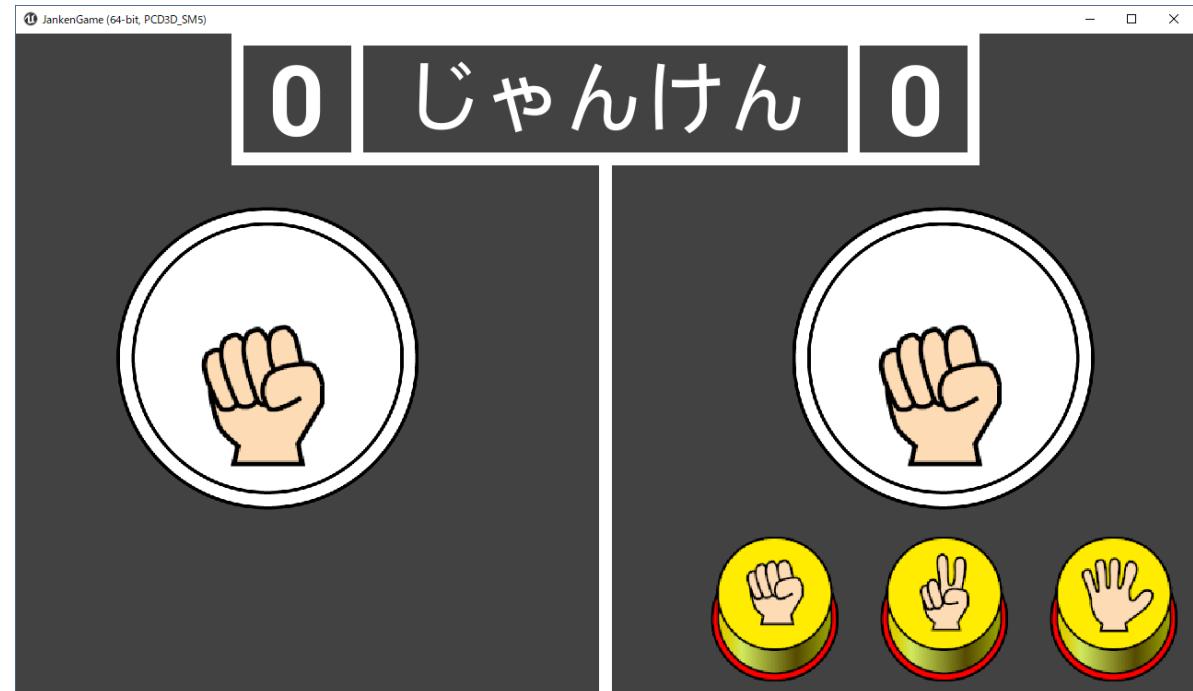


この後に別のブループリントを編集するため、コンパイル > 保存

プレイして確認する



音(BGM)が再生される



13. 音の追加

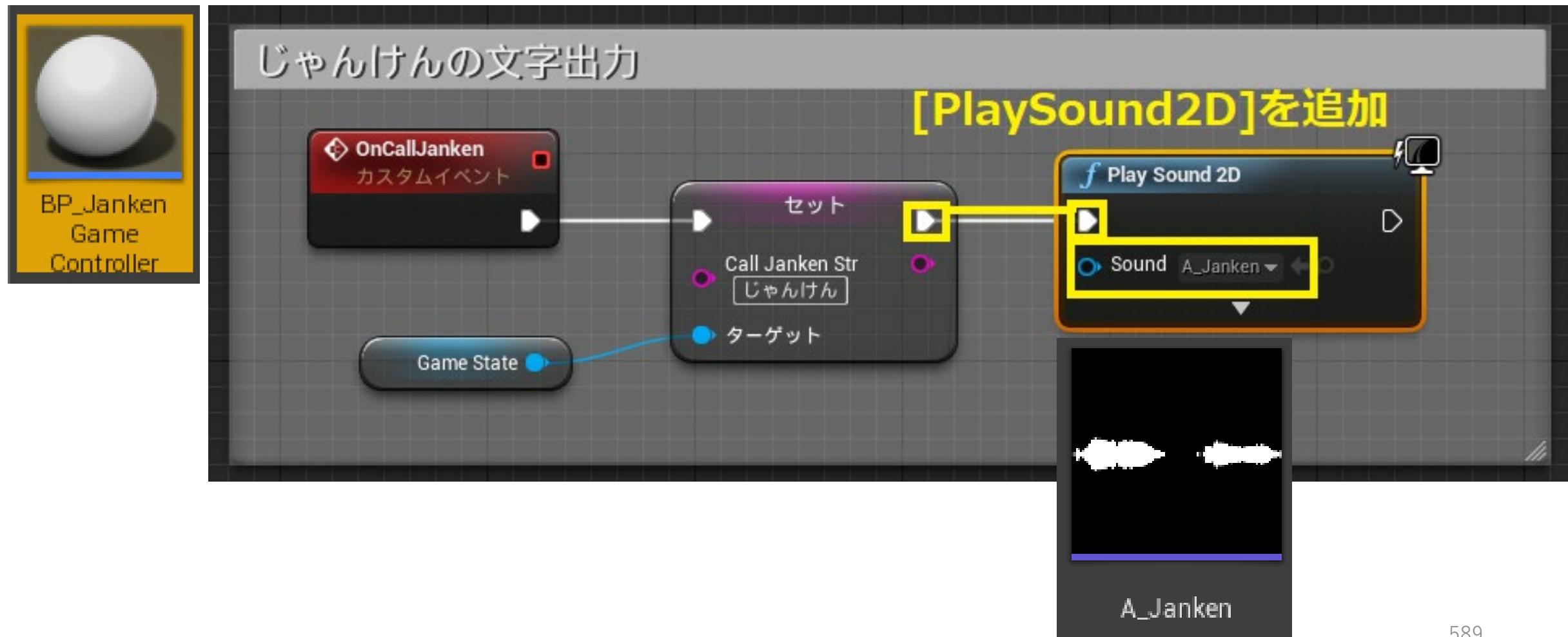
13.1 音データのインポート

13.2 BGMがループするようにキューを作成する

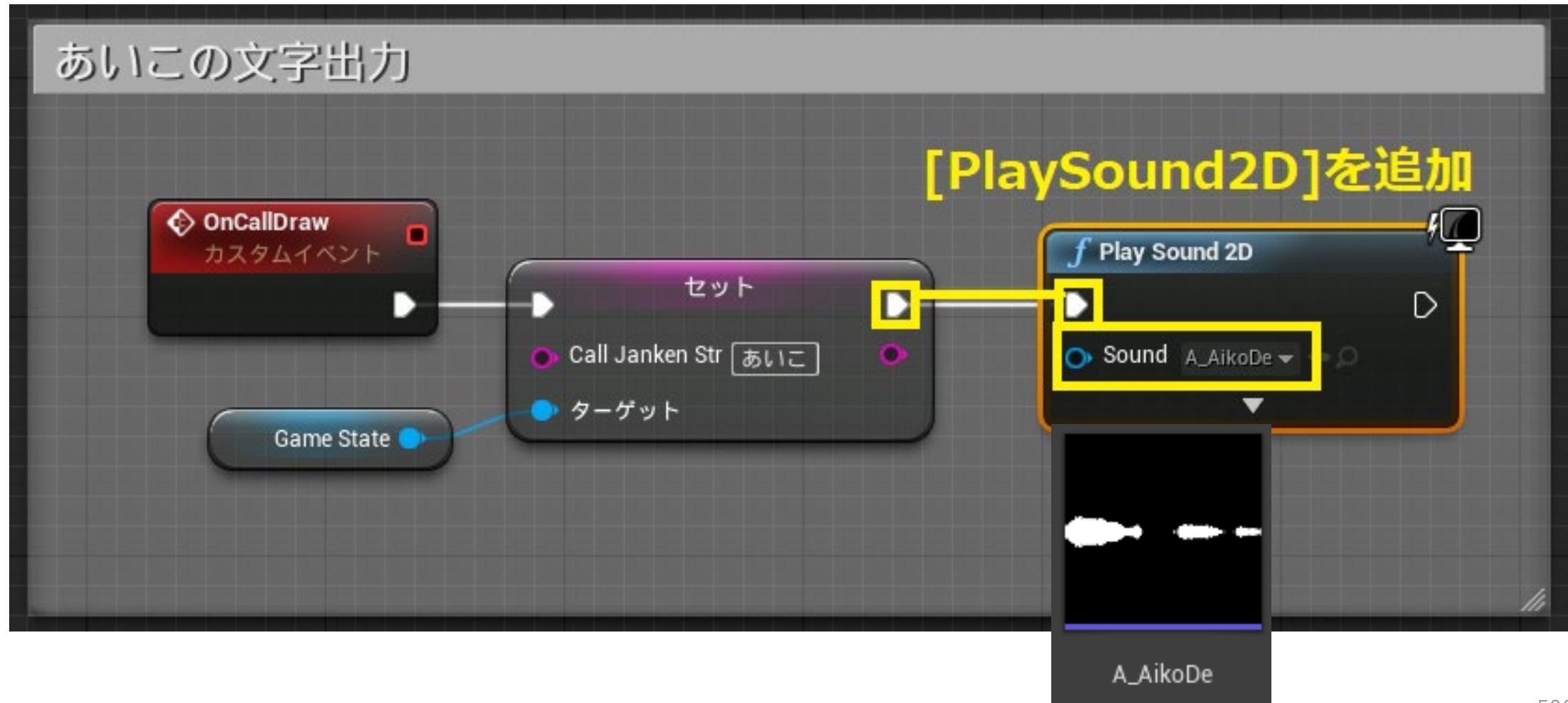
13.3 BGMを再生する

13.4 SEを再生する

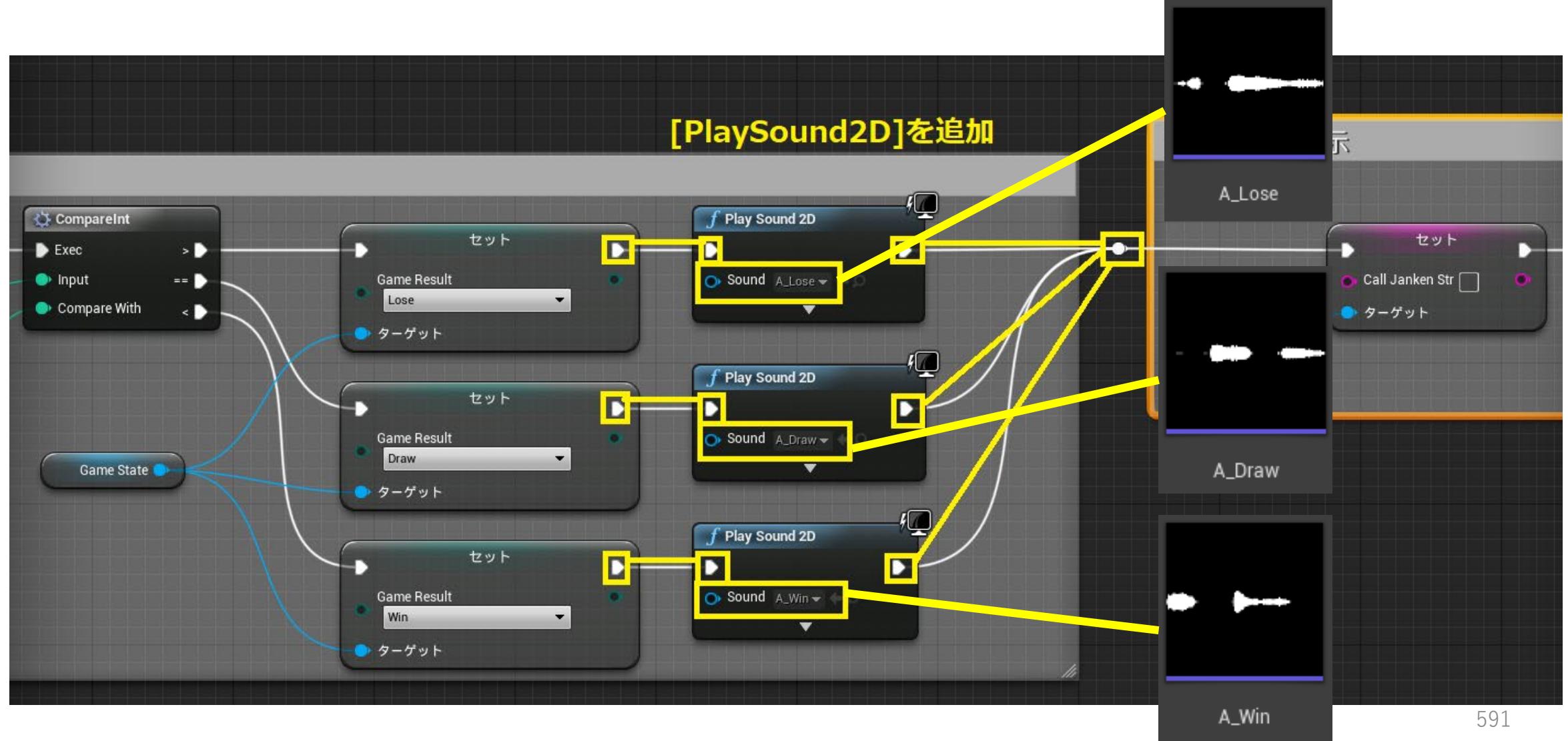
A_Jankenを再生する



A_AikoDeを再生する



勝敗の音(A_Win,A_Lose,A_Draw)を再生する

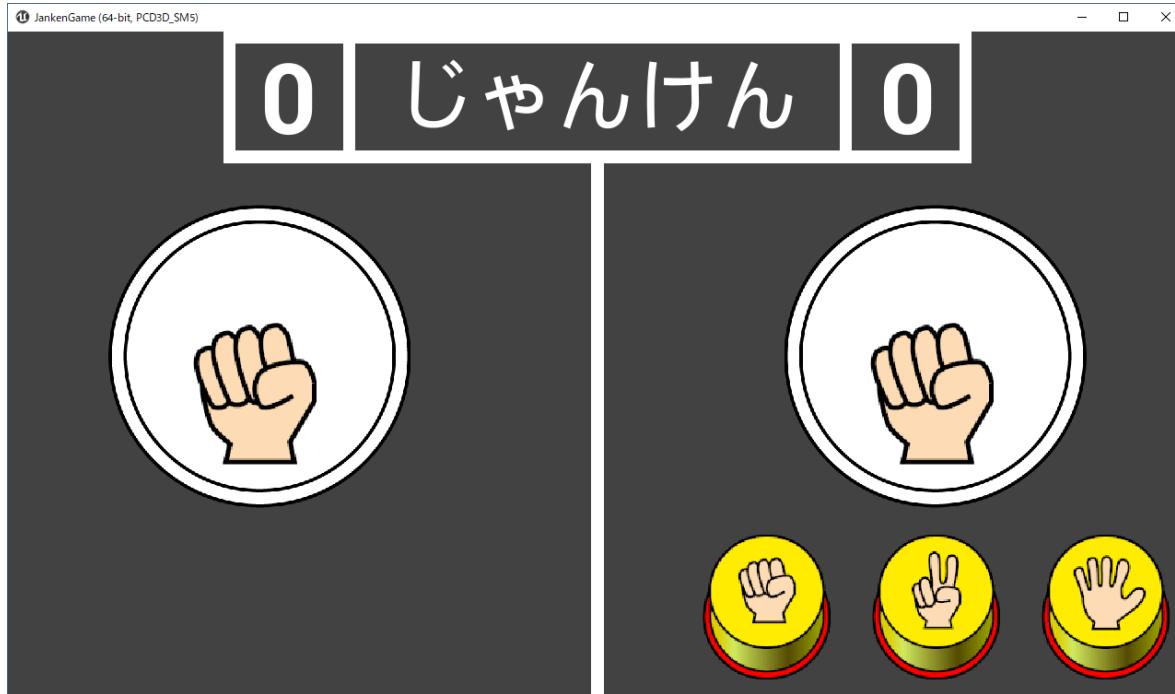


コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

プレイして確認する



音(SE)が再生される

14. 解像度設定

14. 解像度設定

14.1 列挙型 EResolutionを作成

14.2 ComboBoxにEResolutionのエニュミレータをすべて表示

14.3 解像度を変える

14.4 レベルを跨いで値を保持するためにBP_GameInstanceを作成

14.5 解像度の設定を保存して、ゲーム開始時に読み込む

14. 解像度設定

14.1 列挙型 EResolutionを作成

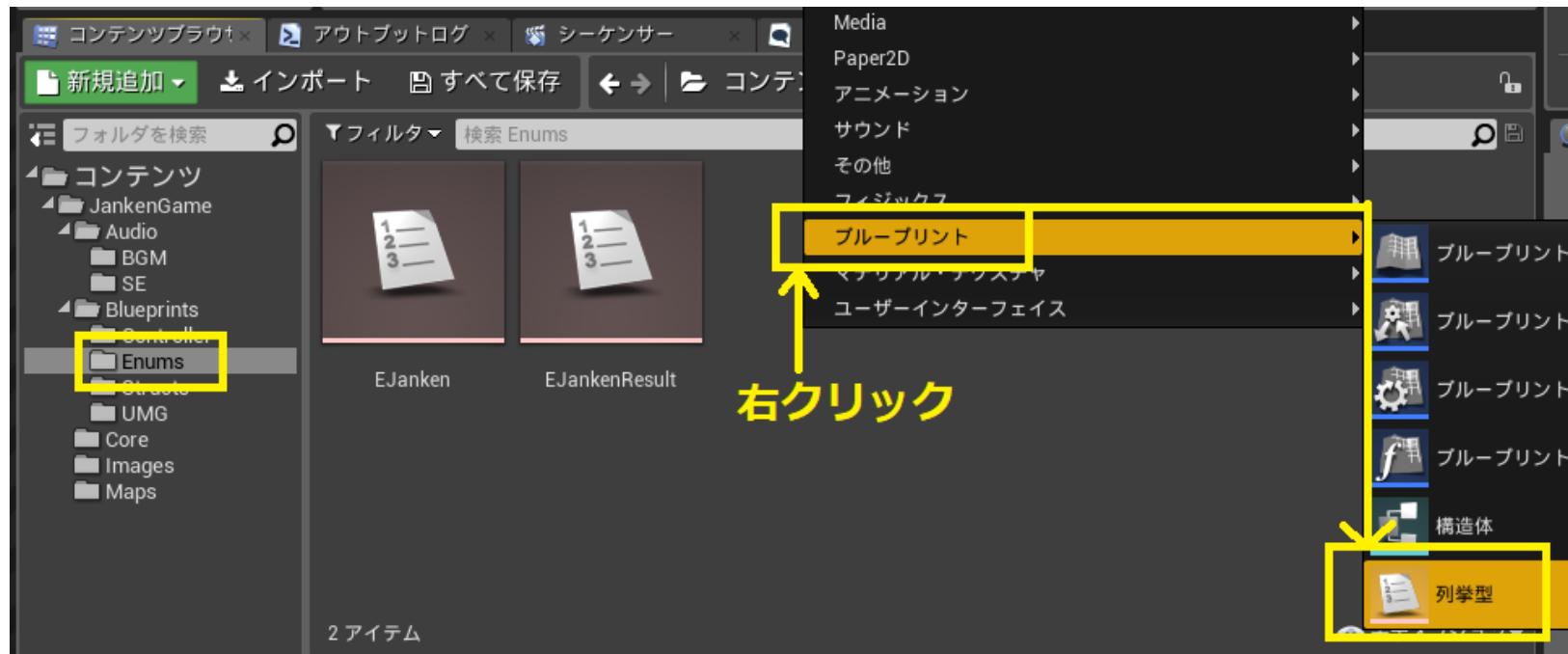
14.2 ComboBoxにEResolutionのエニュミレータをすべて表示

14.3 解像度を変える

14.4 レベルを跨いで値を保持するためにBP_GameInstanceを作成

14.5 解像度の設定を保存して、ゲーム開始時に読み込む

列挙型::EResolutionを作成する 1



列挙型:EResolutionを作成する 2



Descriptionは4.18から追加されたため、
4.18未満のバージョンの場合は設定項目がない

エニュメレーター

Display Name	Description
FullScreen	フルスクリーン
1920x1080	1920x1080(16:9)
1280x720	1280x720(16:9)
960x540	960x540(16:9)

Description

Enum Description

ゲームの解像度

14. 解像度設定

14.1 列挙型 EResolutionを作成

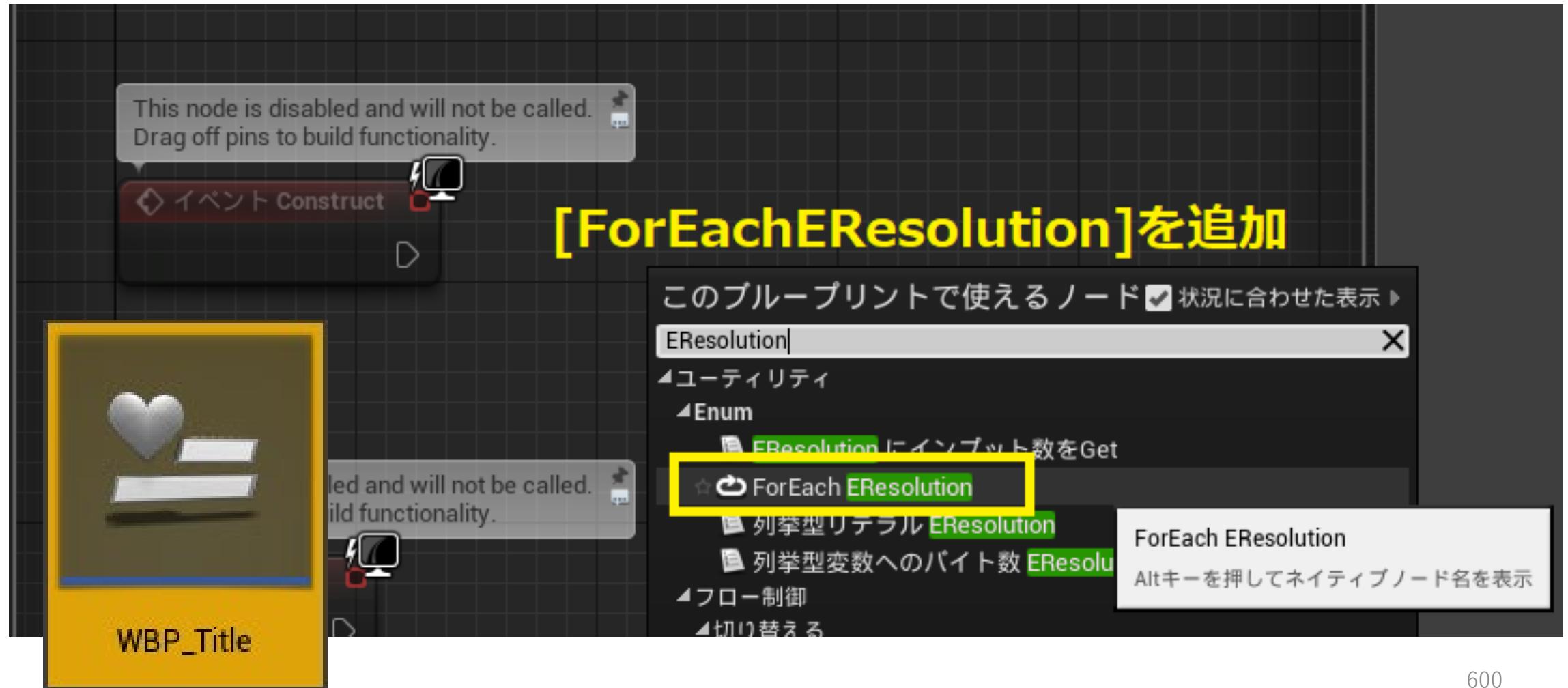
14.2 ComboBoxにEResolutionのエニュミレータをすべて表示

14.3 解像度を変える

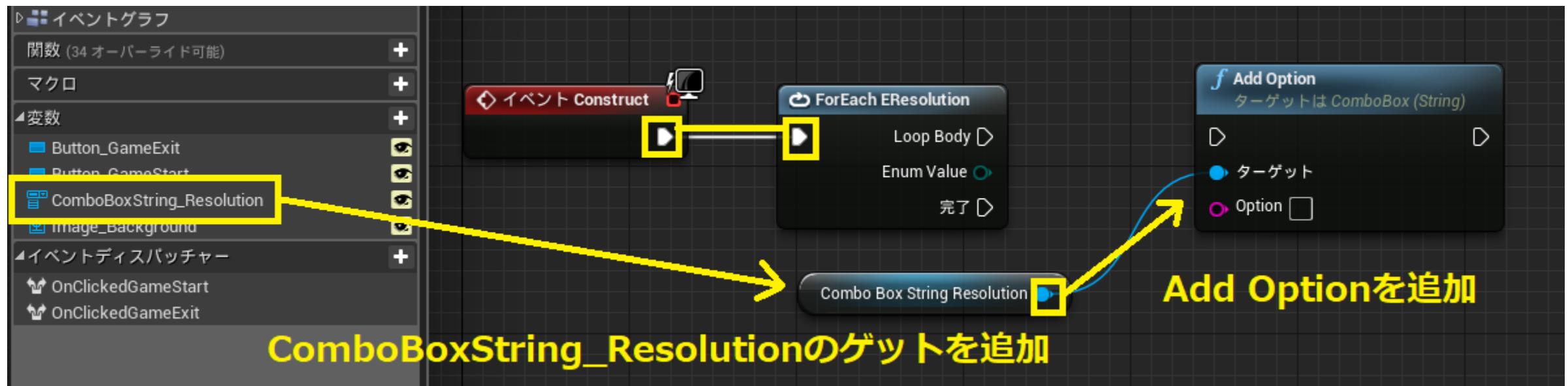
14.4 レベルを跨いで値を保持するためにBP_GameInstanceを作成

14.5 解像度の設定を保存して、ゲーム開始時に読み込む

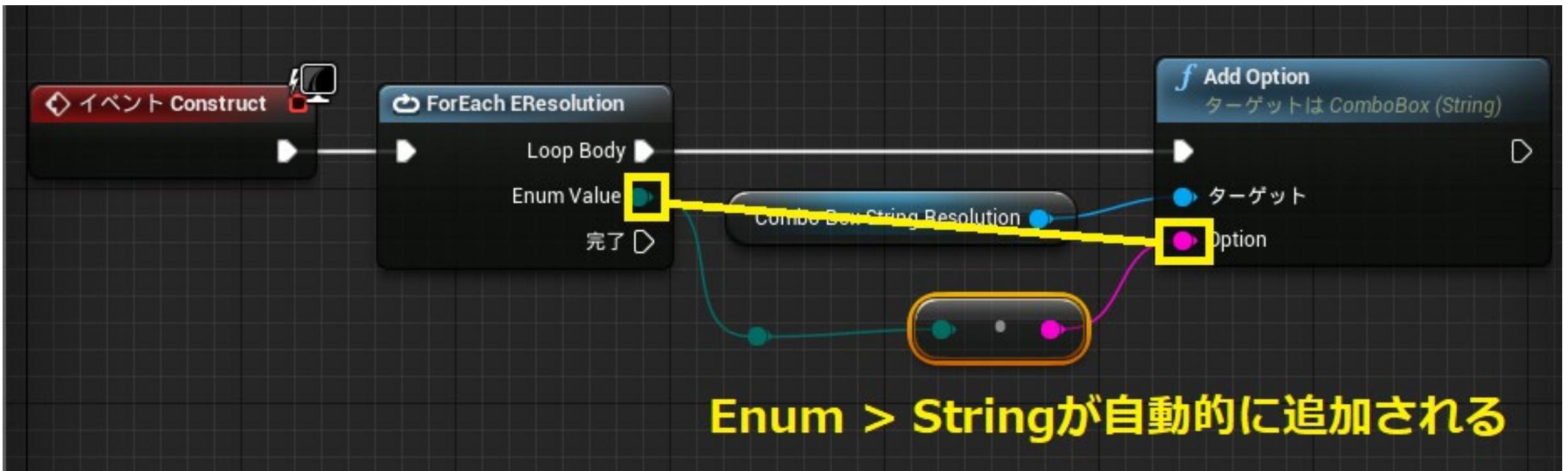
イベント Construct の処理を実装する 1



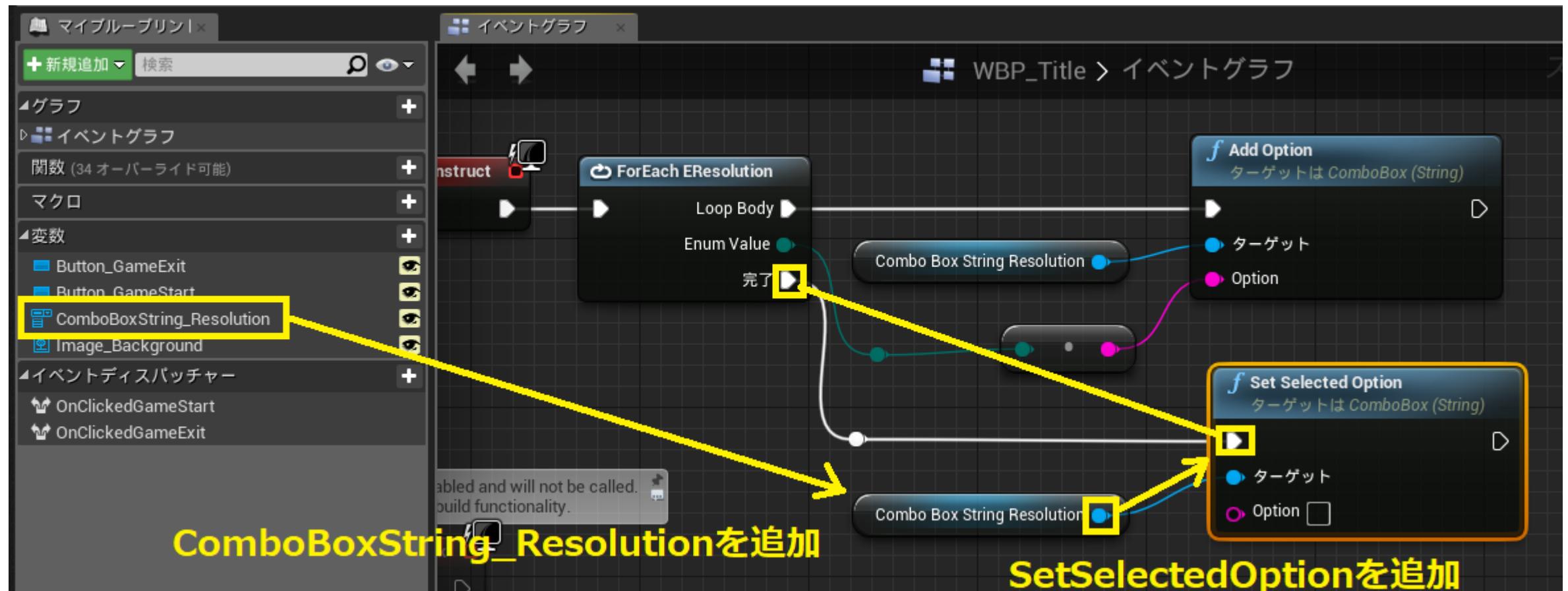
イベント Construct の処理を実装する 2



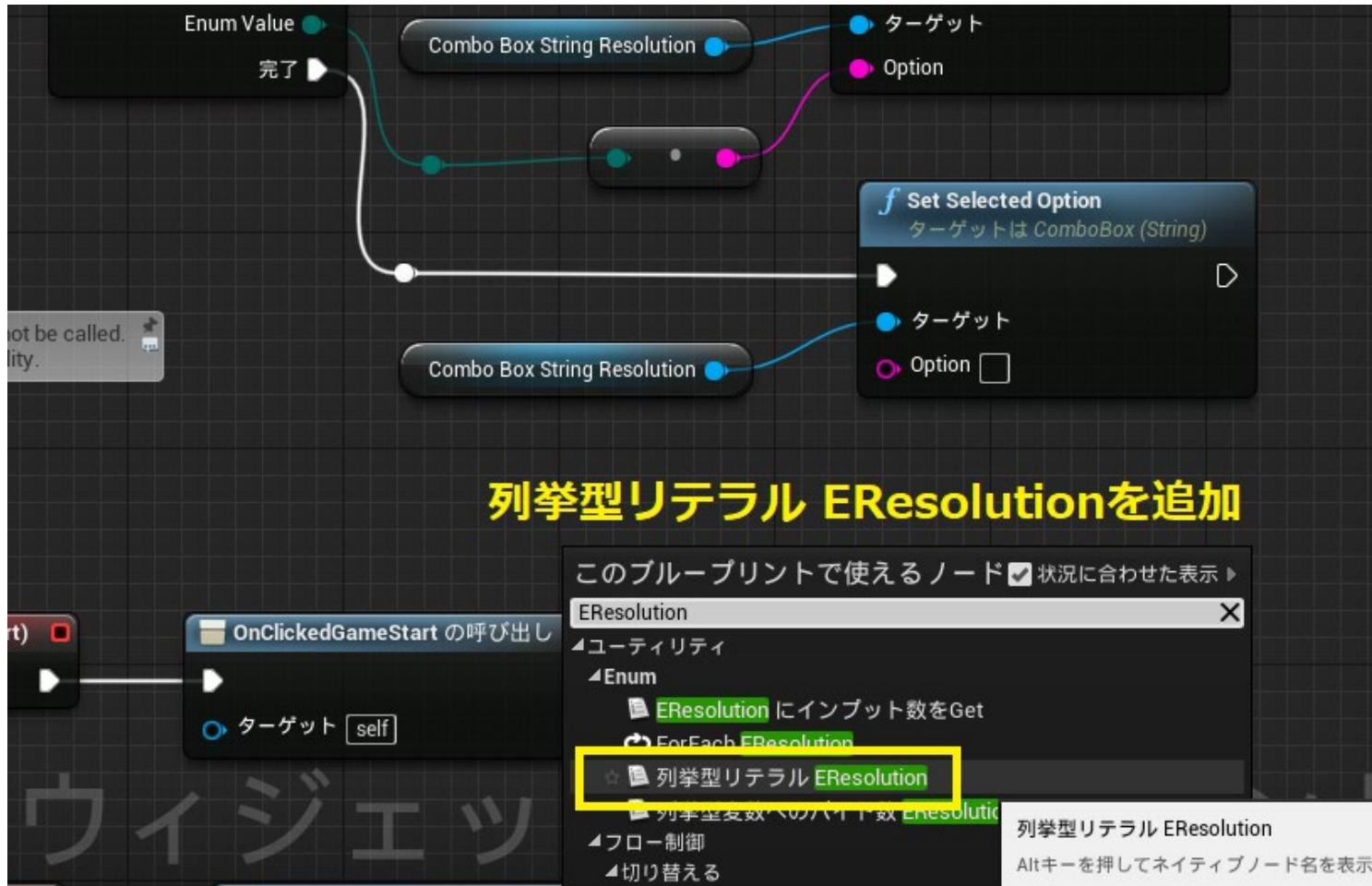
イベント Construct の処理を実装する 3



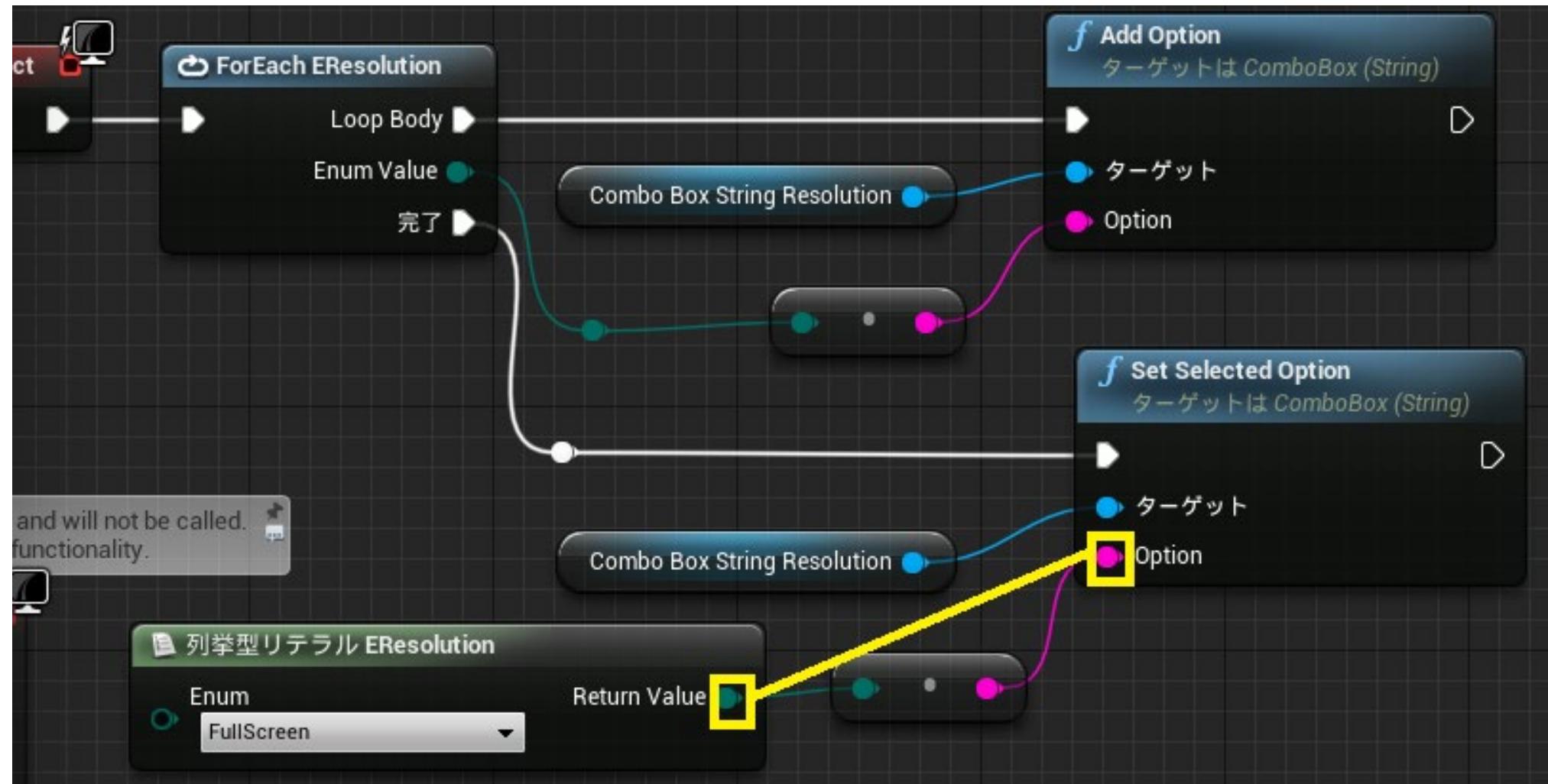
イベントConstructの処理を実装する 4



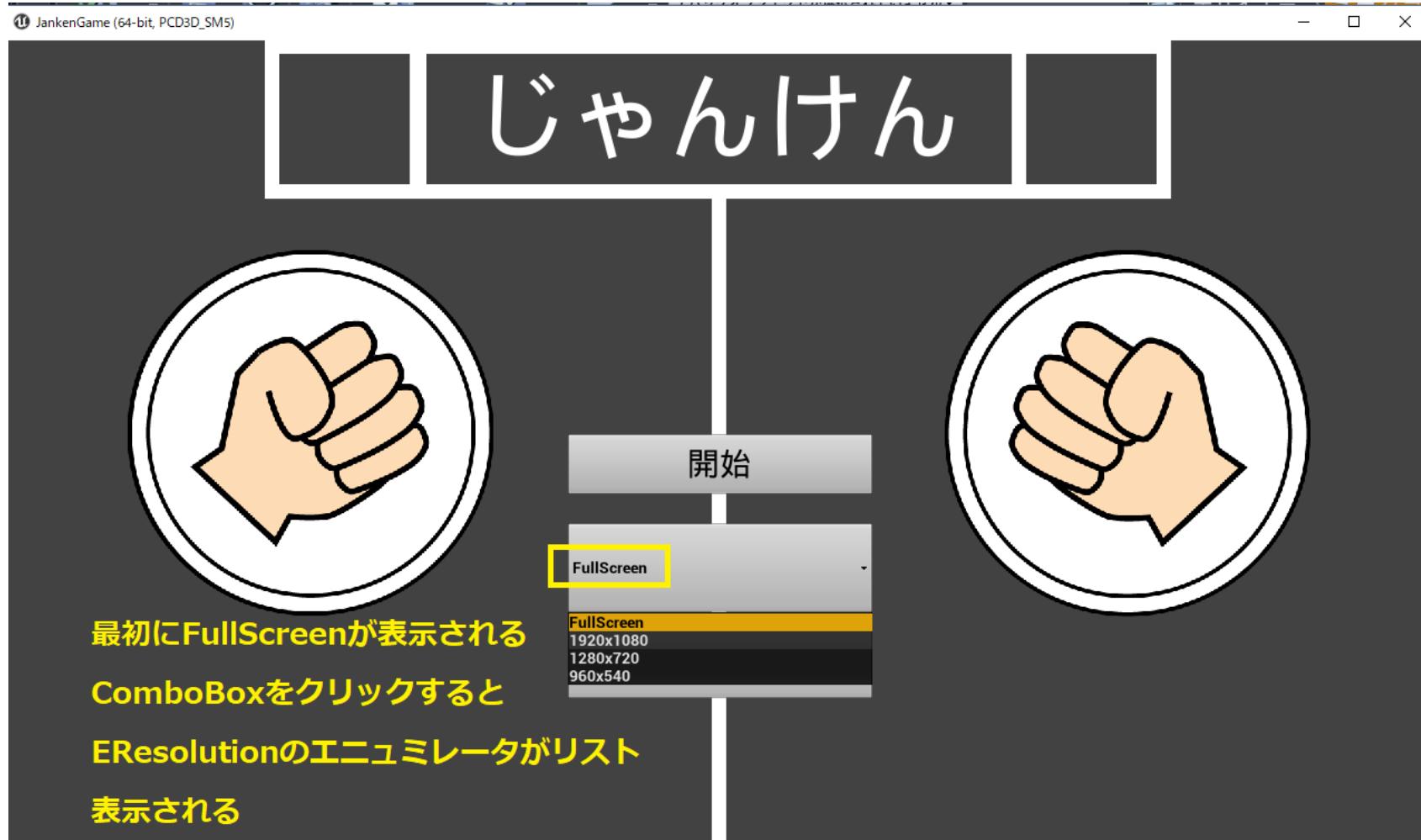
イベント Construct の処理を実装する 4



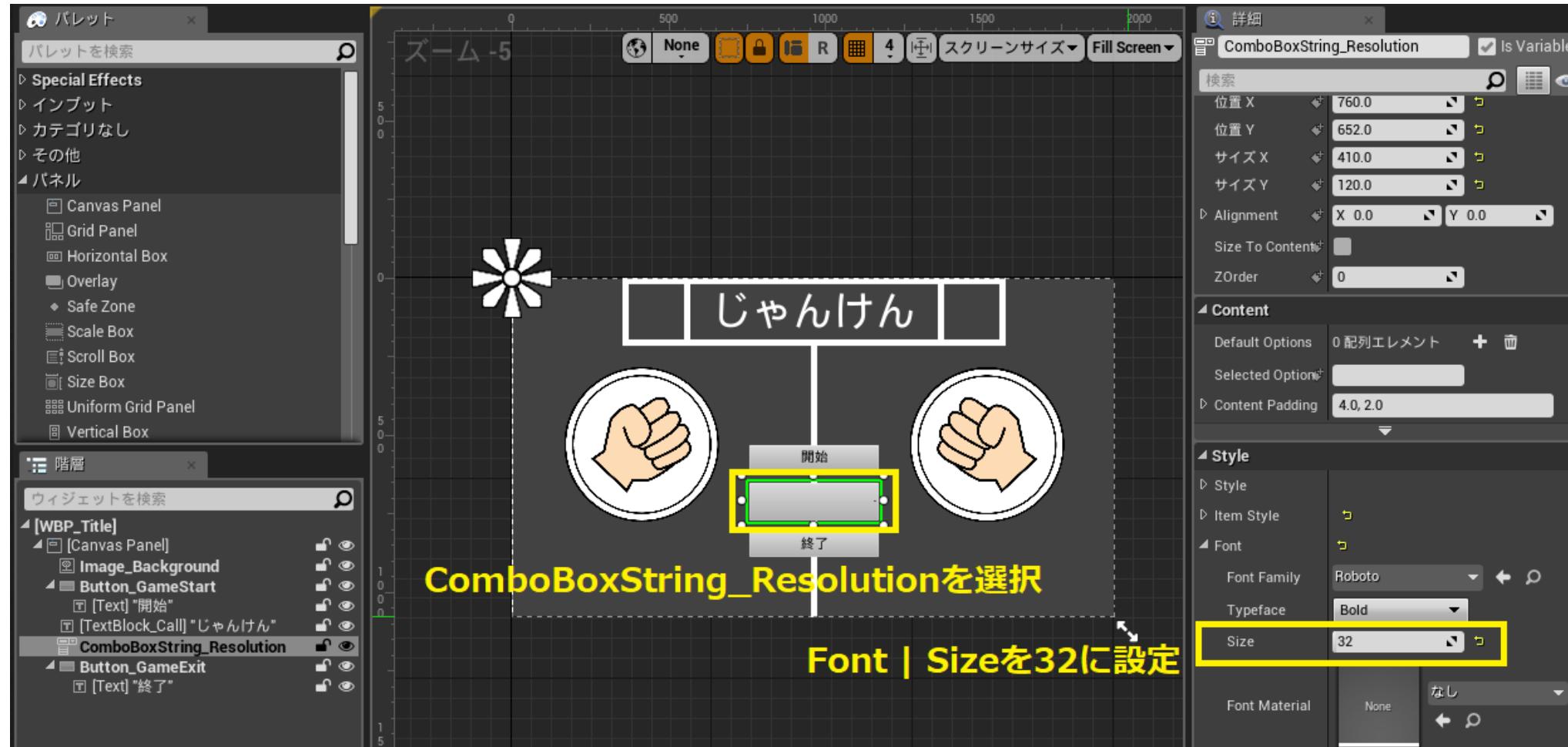
イベント Construct の処理を実装する 5



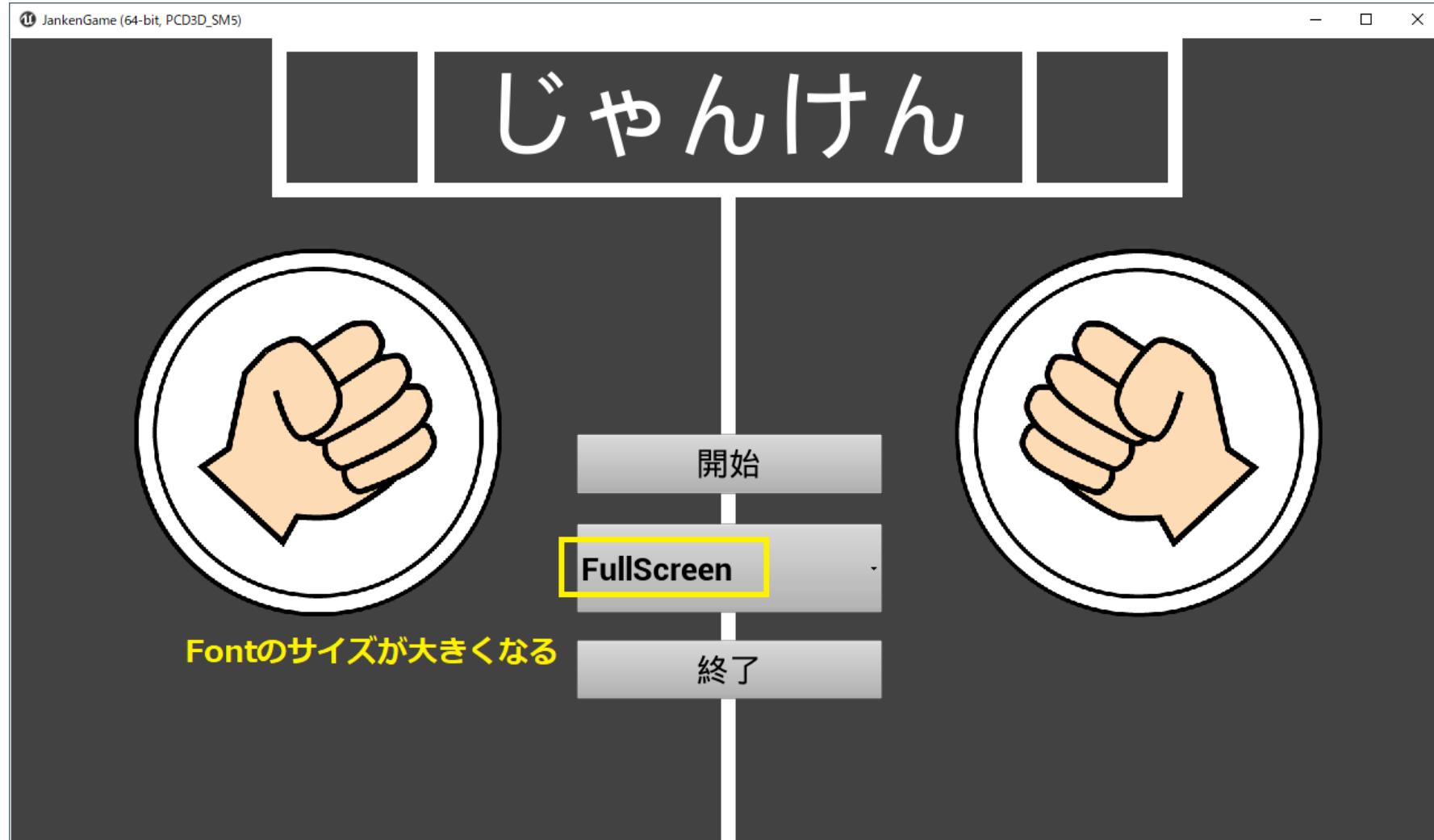
プレイして確認する



ComboBoxStringのFont Sizeを変更する



プレイして確認する



14. 解像度設定

14.1 列挙型 EResolutionを作成

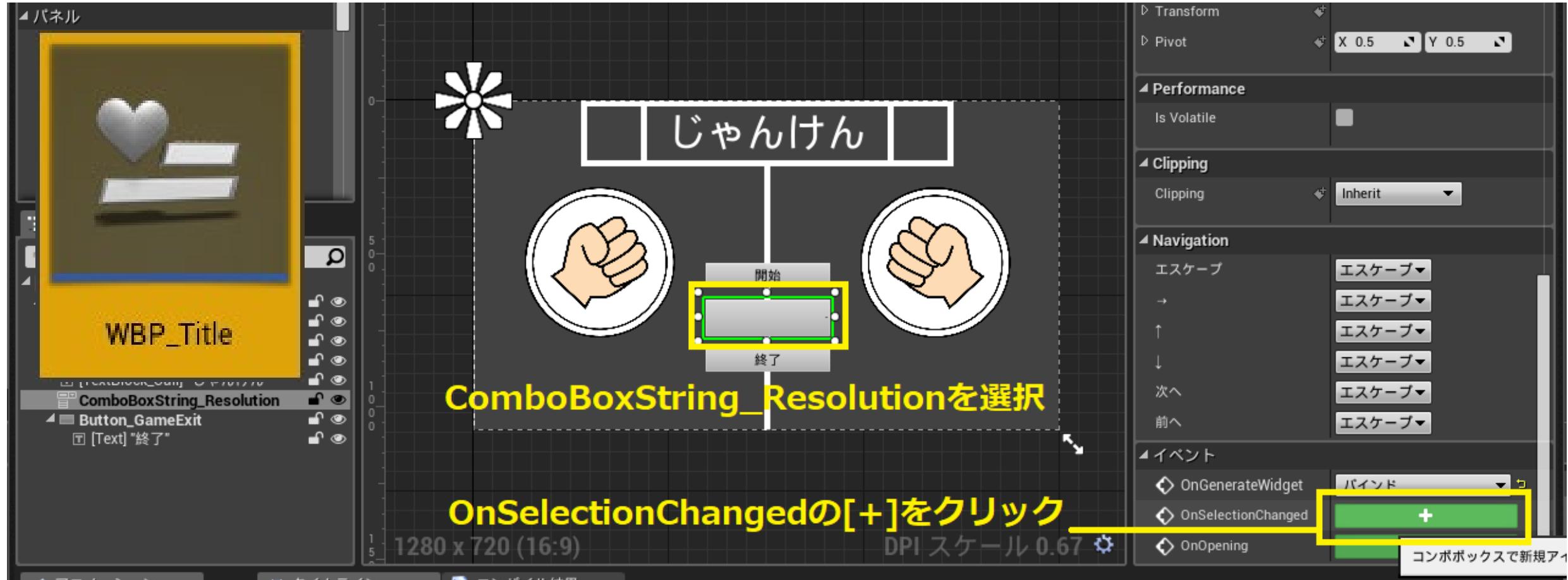
14.2 ComboBoxにEResolutionのエニュミレータをすべて表示

14.3 解像度を変える

14.4 レベルを跨いで値を保持するためにBP_GameInstanceを作成

14.5 解像度の設定を保存して、ゲーム開始時に読み込む

ComboBoxString_Resolutionに OnSelectionChangedイベントを追加する 1



ComboBoxString_Resolutionに OnSelectionChangedイベントを追加する 2

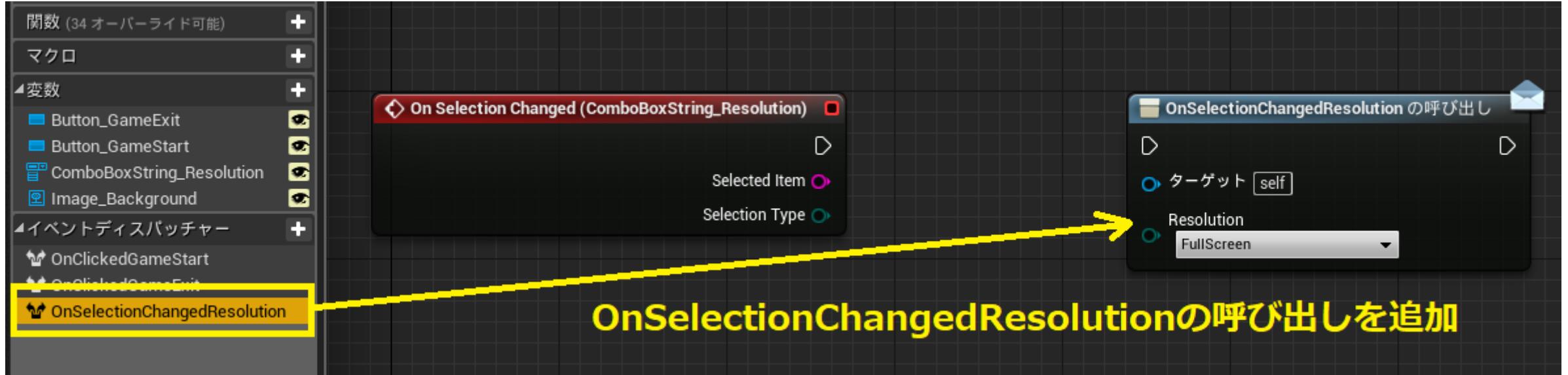


名前を[OnSelectionChangedResolution]に設定



インプット パラメータ名	Description
Resolution	EResolution

ComboBoxString_Resolutionに OnSelectionChangedイベントを追加する 3



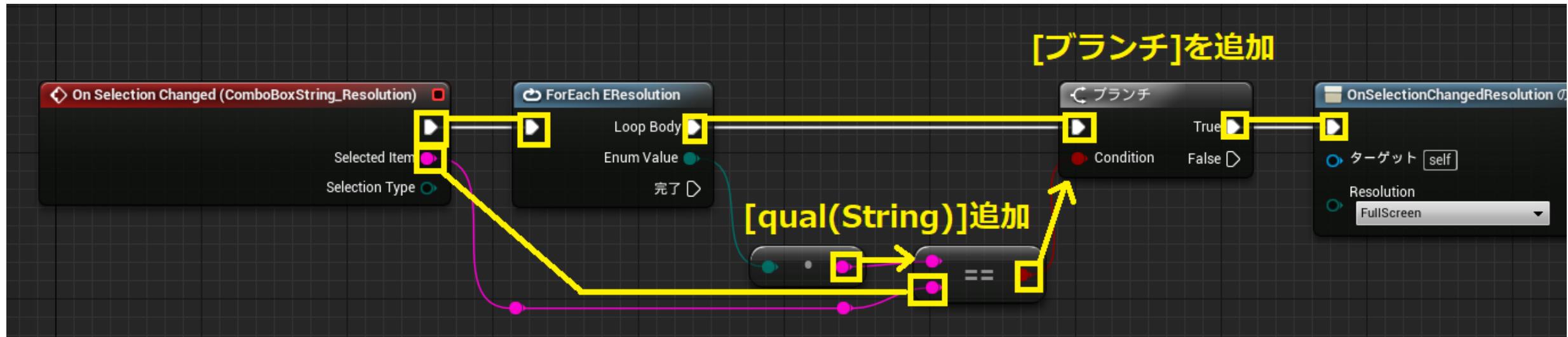
ComboBoxString_Resolutionに OnSelectionChangedイベントを追加する 4



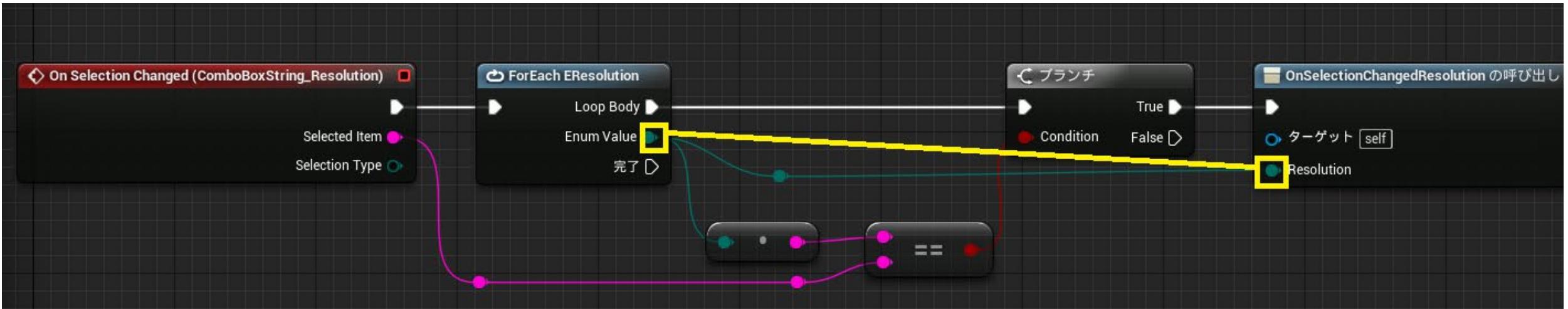
ComboBoxString_Resolutionに OnSelectionChangedイベントを追加する 5



ComboBoxString_Resolutionに OnSelectionChangedイベントを追加する 6



ComboBoxString_Resolutionに OnSelectionChangedイベントを追加する 7

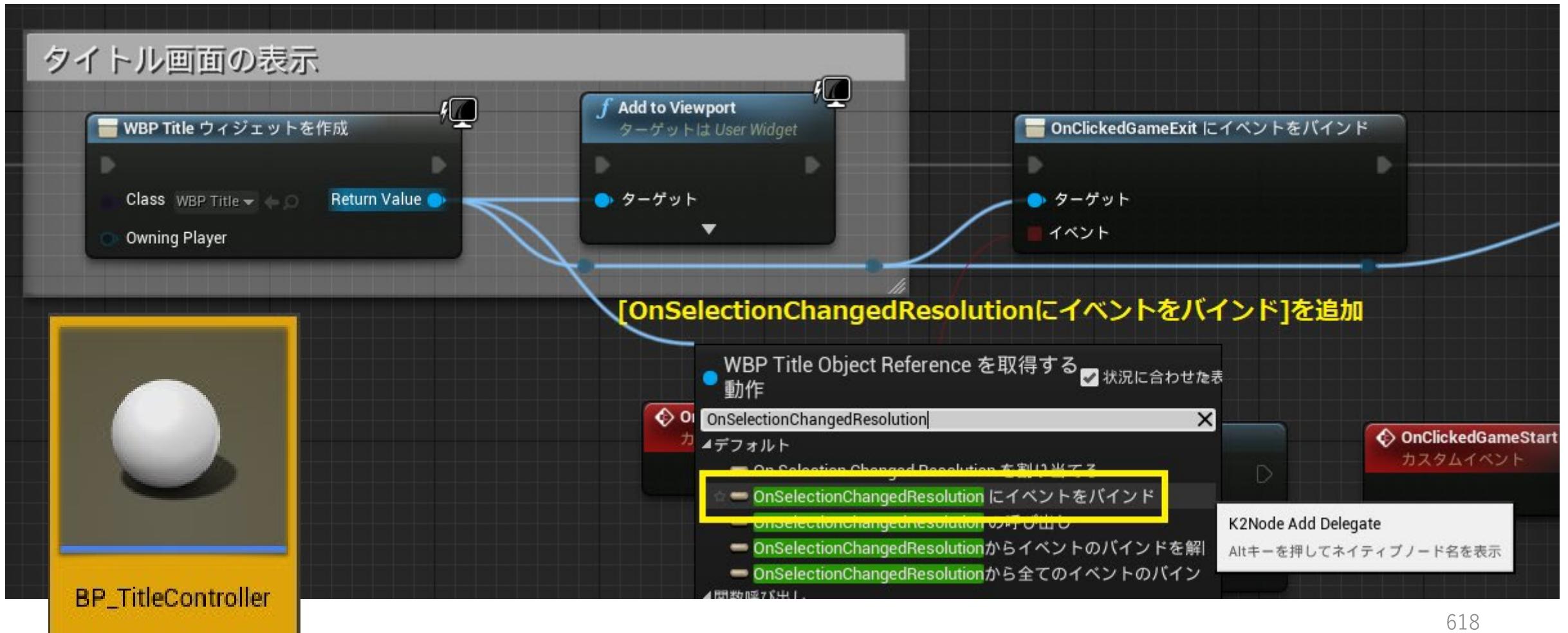


コンパイル > 保存

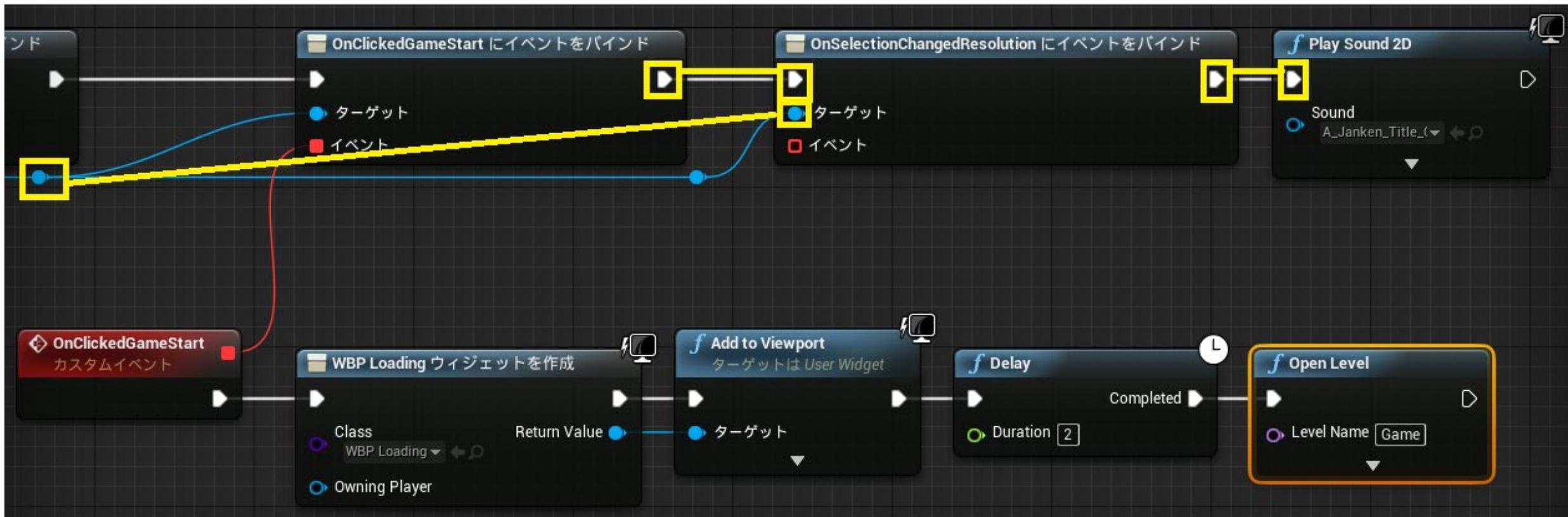


この後に別のブループリントを編集するため、コンパイル > 保存

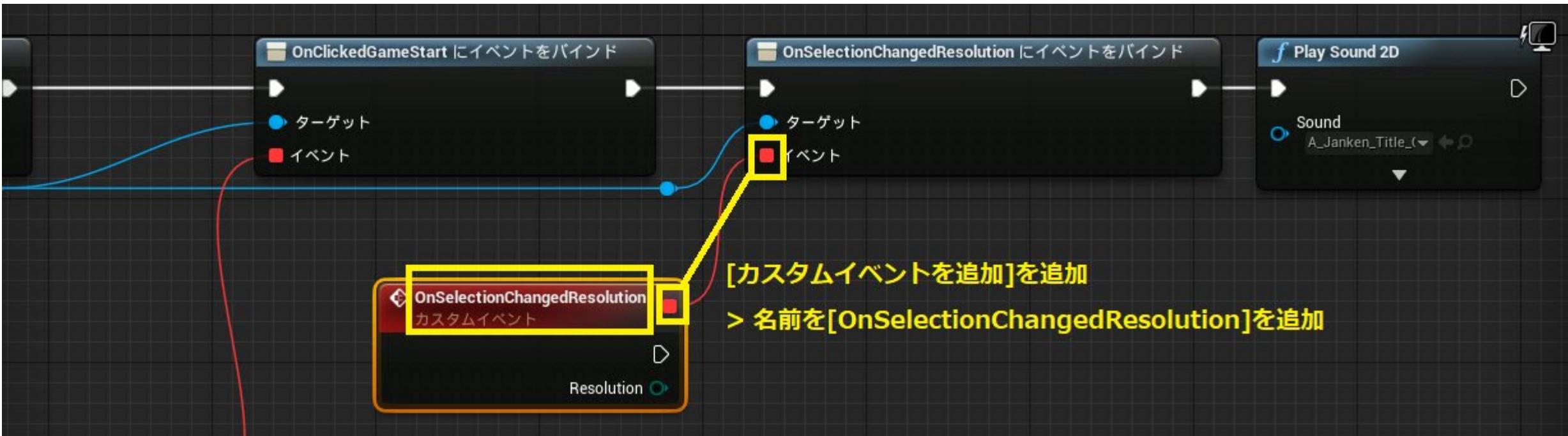
OnSelectionChangedResolutionにイベントをバインドする 1



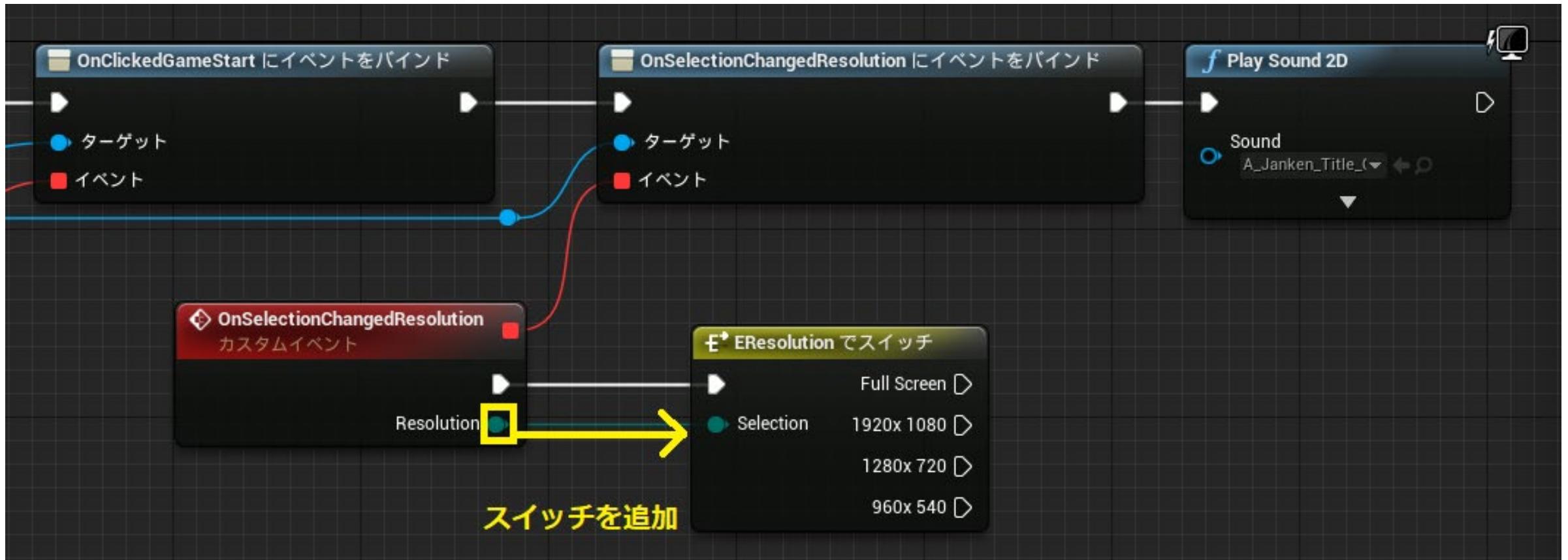
OnSelectionChangedResolutionにイベントをバインドする 2



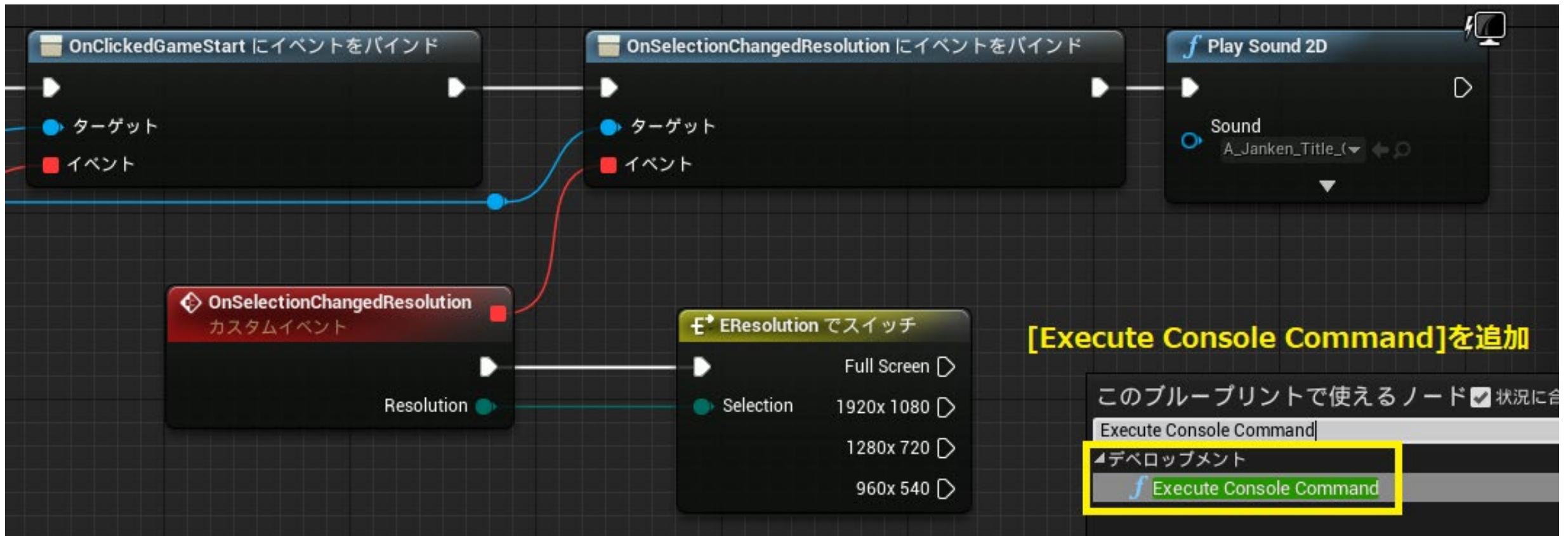
OnSelectionChangedResolutionにイベントをバインドする 3



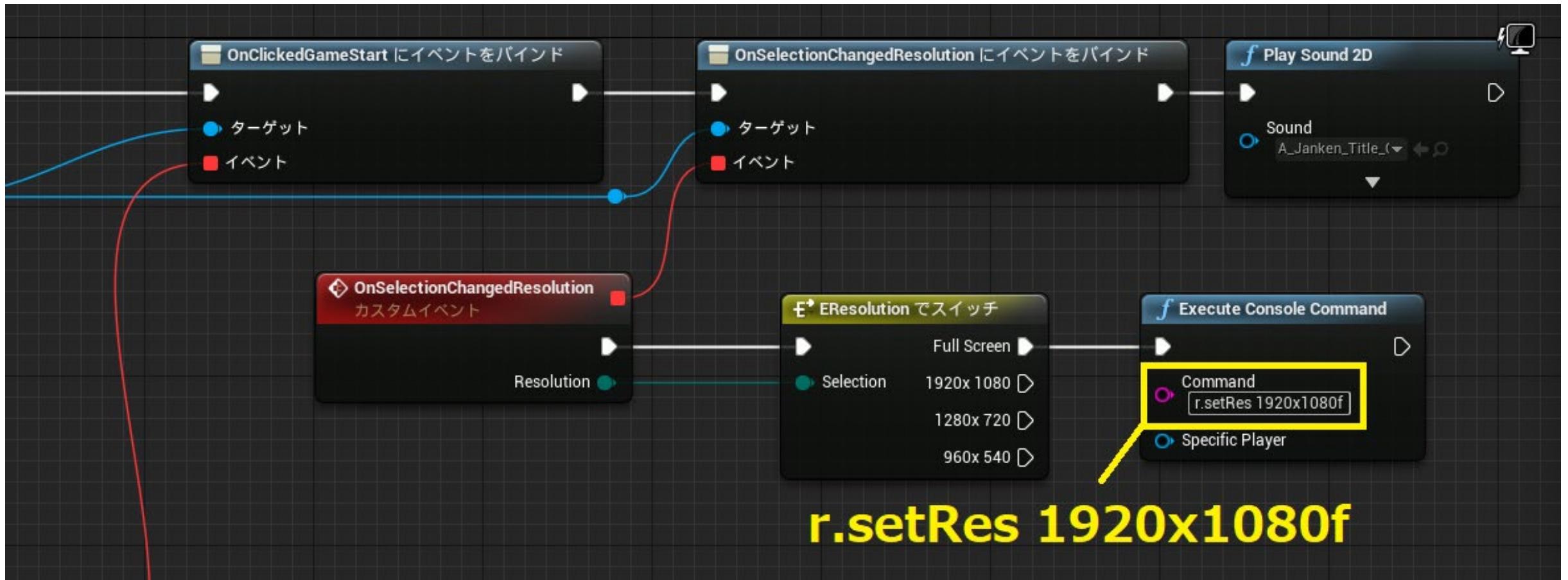
OnSelectionChangedResolutionにイベントをバインドする 4



OnSelectionChangedResolutionにイベントをバインドする 5

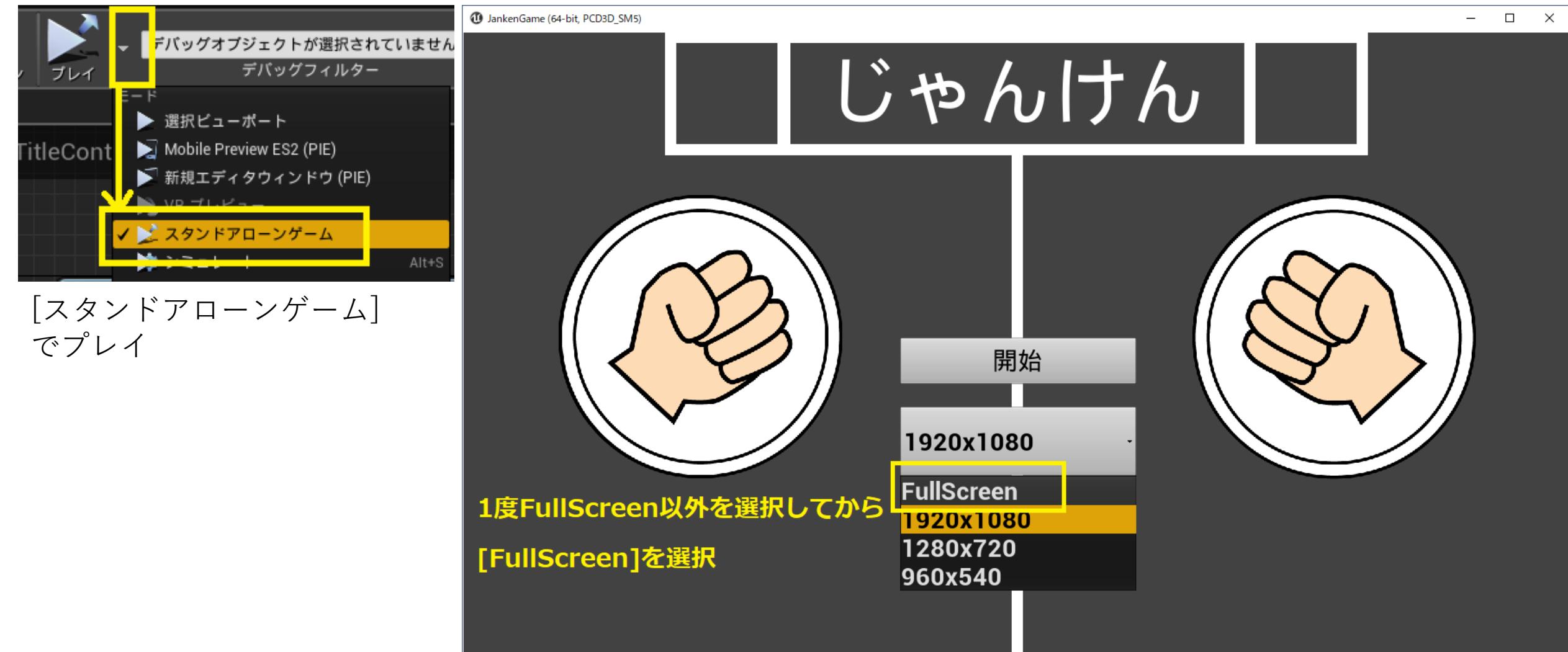


OnSelectionChangedResolutionにイベントをバインドする 6

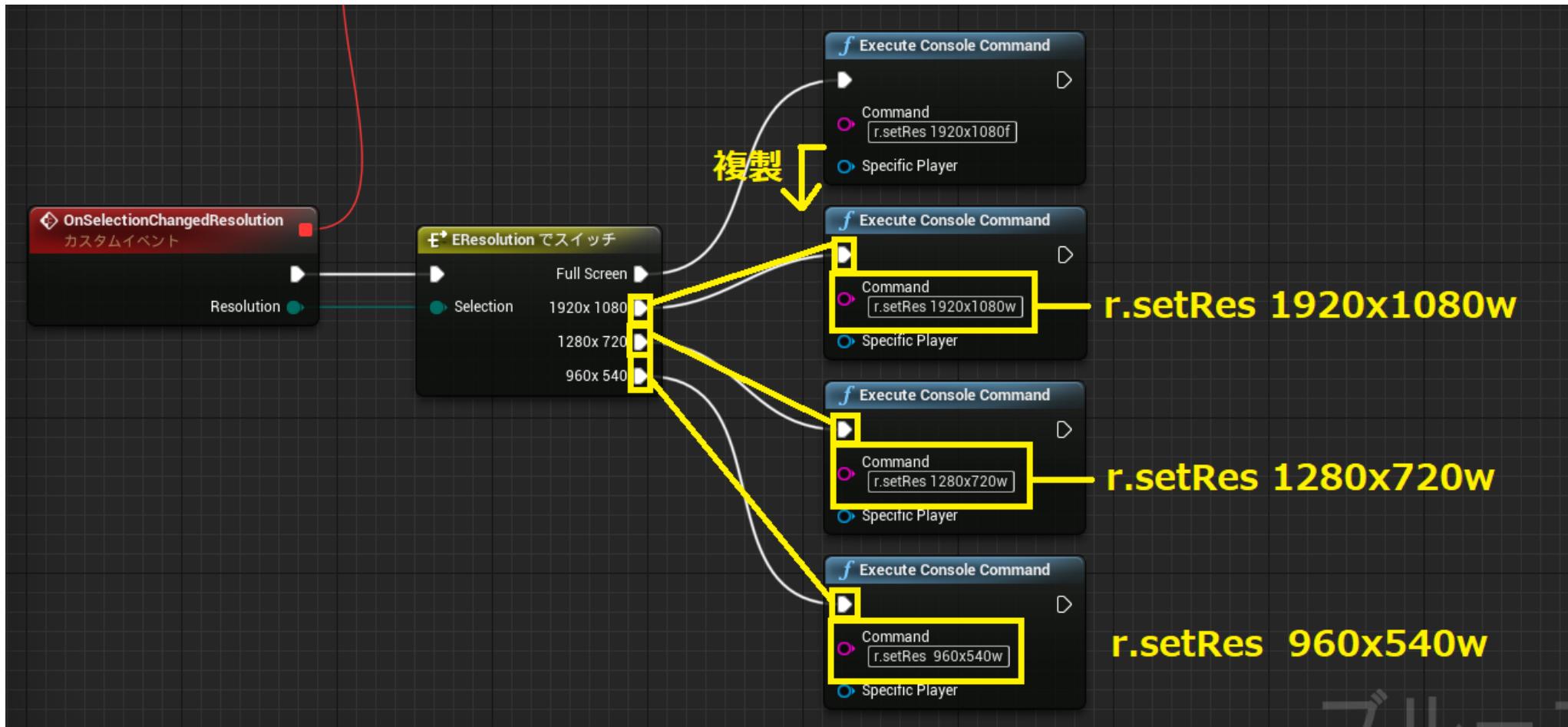


r.setRes 1920x1080f

スタンドアローンゲームで確認する



OnSelectionChangedResolutionにイベントをバインドする 7

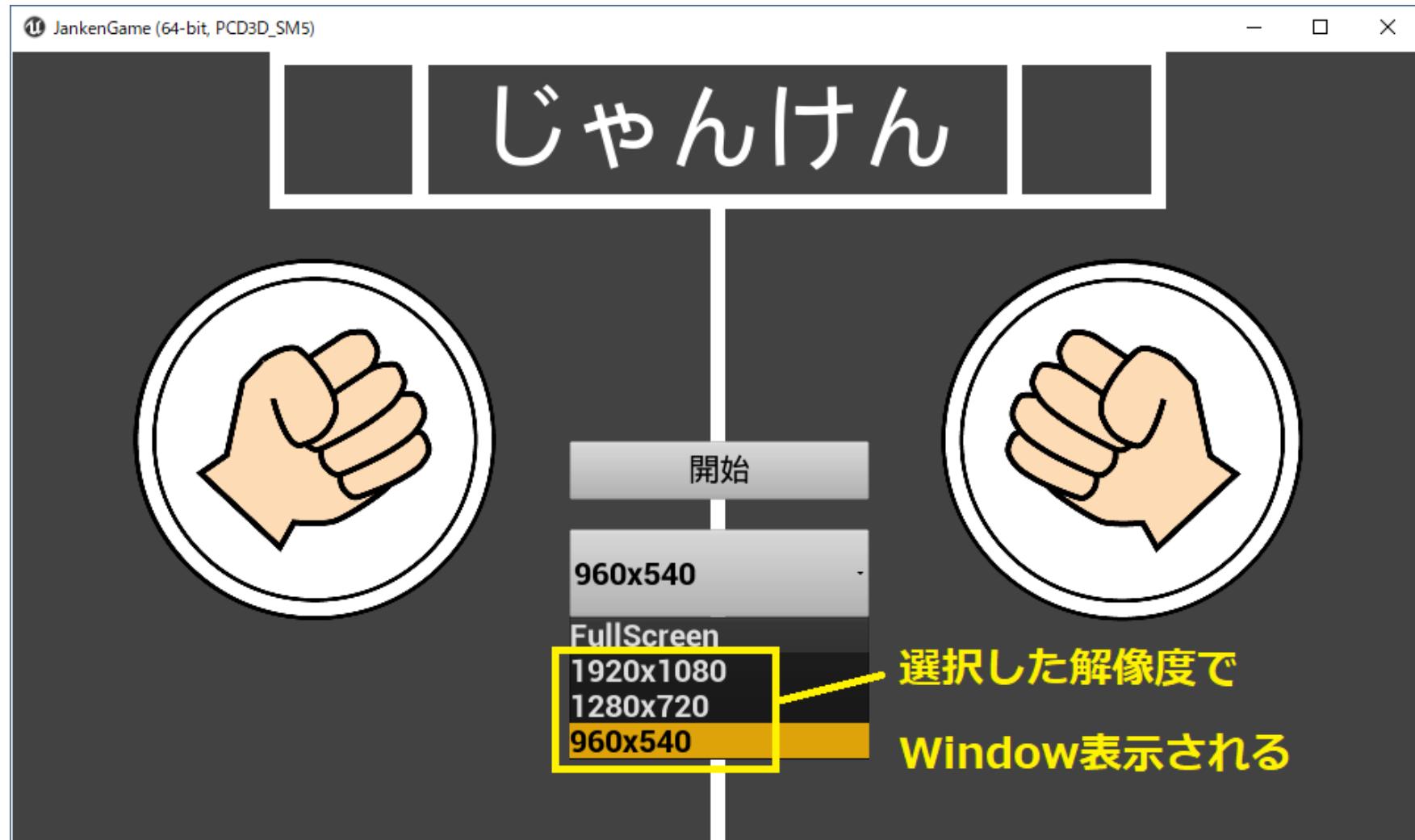


コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

プレイして確認する



解像度の設定コマンド

r.setRes (幅) x (高さ) (f/w)

f:フルスクリーン
w:ウィンドウ

フルスクリーンで1920x1080

r.setRes 1920x1080f

ウィンドウで1920x1080

r.setRes 1920x1080w

14. 解像度設定

14.1 列挙型 EResolutionを作成

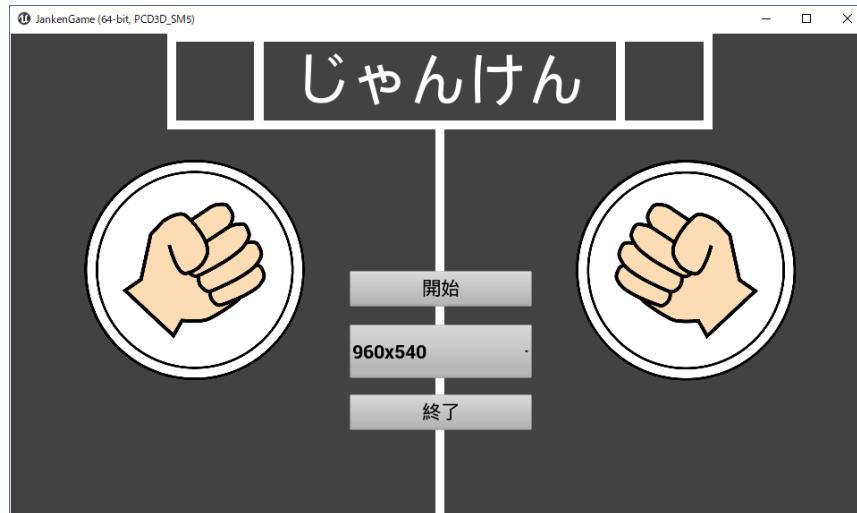
14.2 ComboBoxにEResolutionのエニュミレータをすべて表示

14.3 解像度を変える

14.4 レベルを跨いで値を保持するためにBP_GameInstanceを作成

14.5 解像度の設定を保存して、ゲーム開始時に読み込む

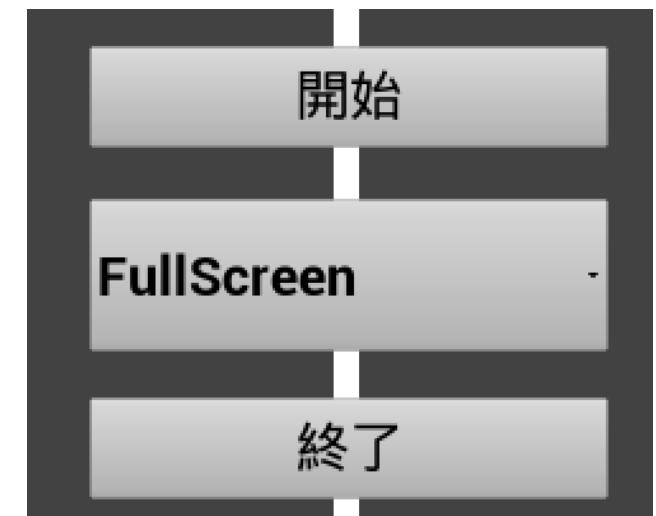
ゲームからタイトルに移動すると解像度がFullScreenに戻ってしまう



解像度を960x540に変更して、開始



GameレベルからTitleレベルに遷移



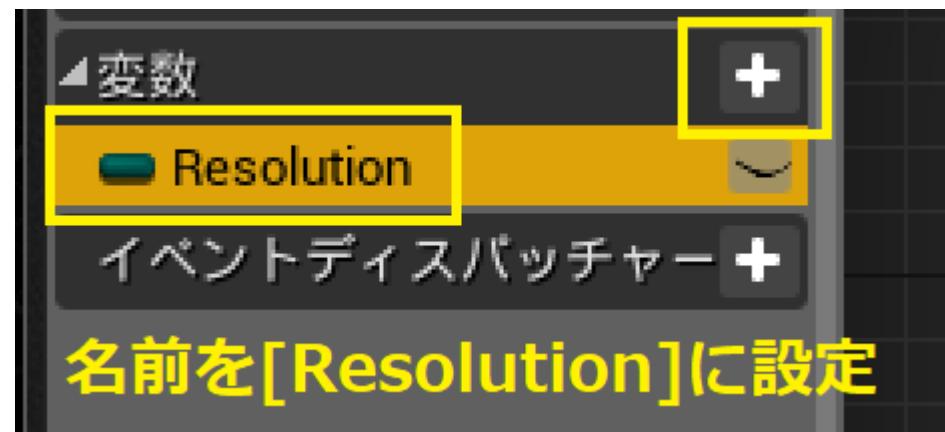
FullScreenに戻っている

BP_GameInstance(親 : GameInstance)を作成する



名前を[BP_GameInstance]に設定

変数の追加する



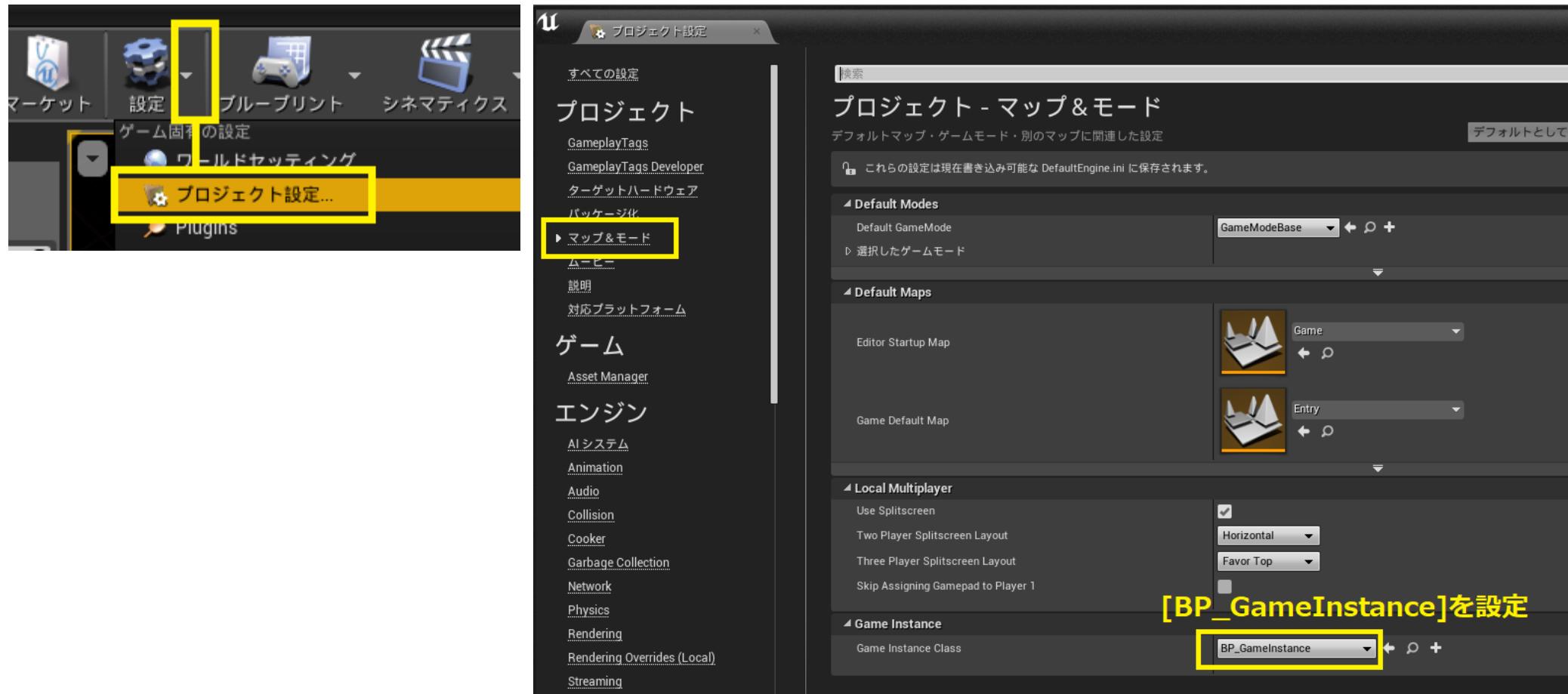
変数名	変数の型	デフォルト値
Resolution	EResolution	-(設定なし)

コンパイル > 保存

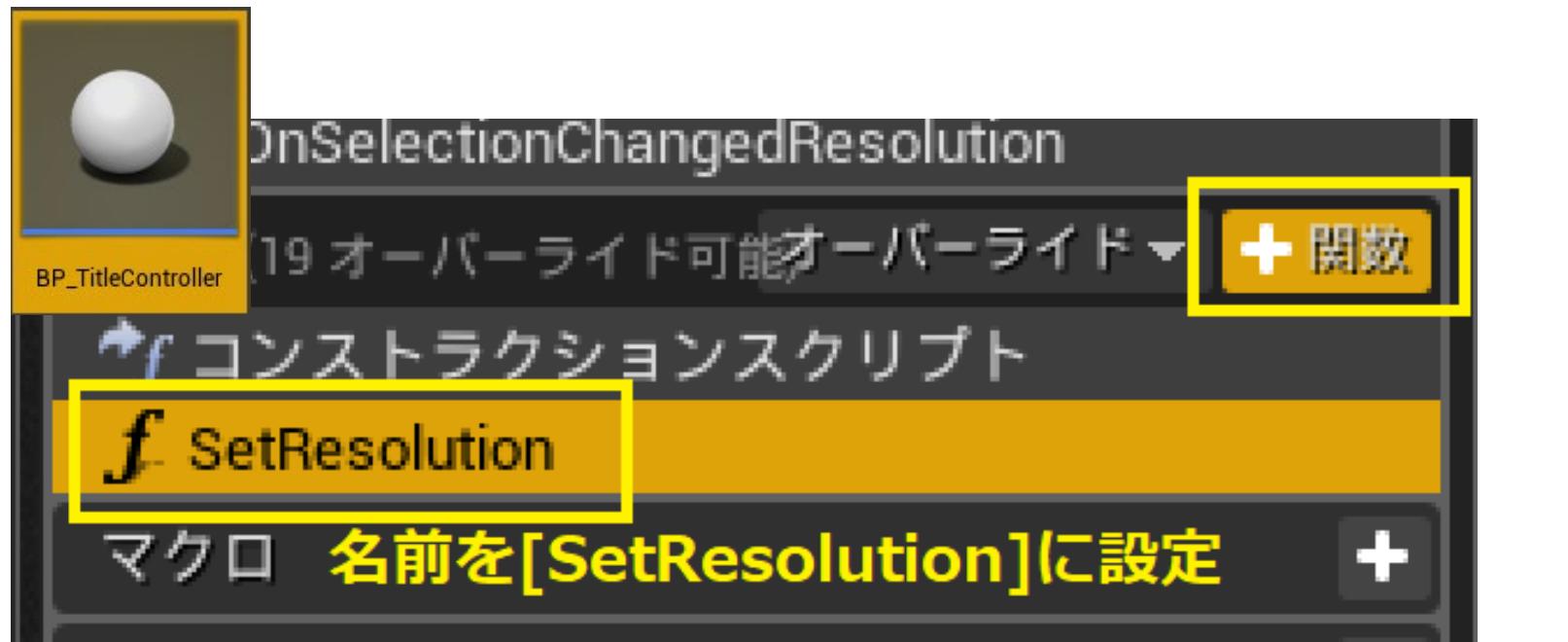


この後に別のブループリントを編集するため、コンパイル > 保存

プロジェクト設定で、GameInstanceClassに BP_GameInstanceを設定する



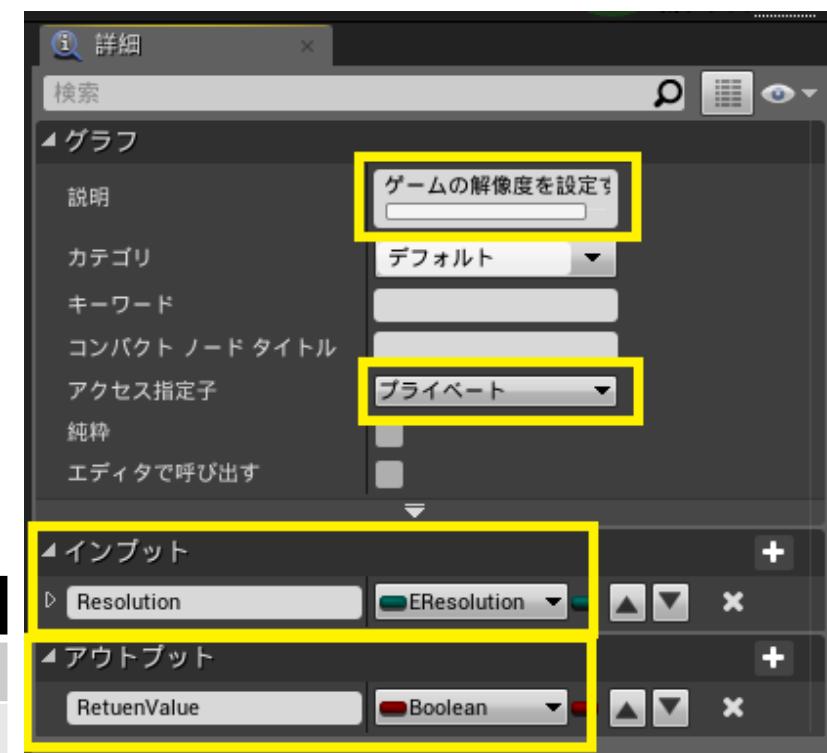
BP_TitleControllerに関数:SetResolutionを追加する 1



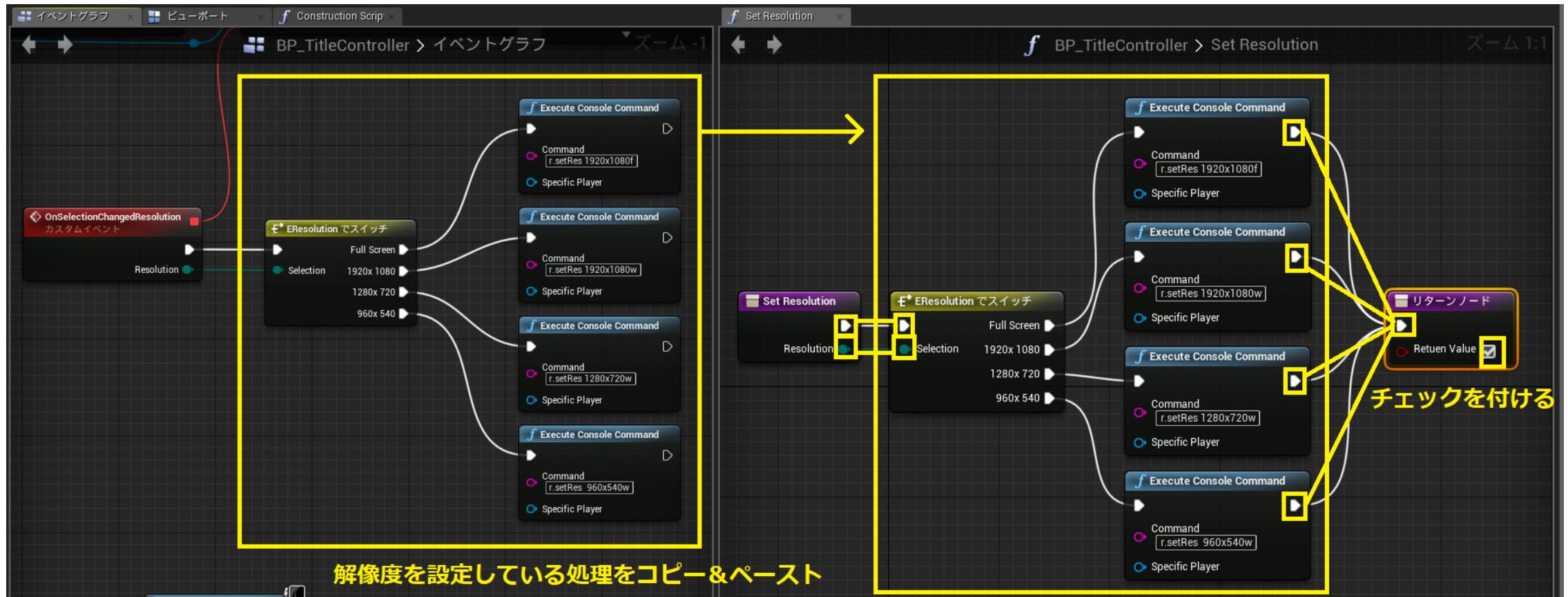
関数名	アクセス指定子	Input/Output	パラメータ名	型
SetResolution	プライベート	Input	Resolution	EResolution

説明

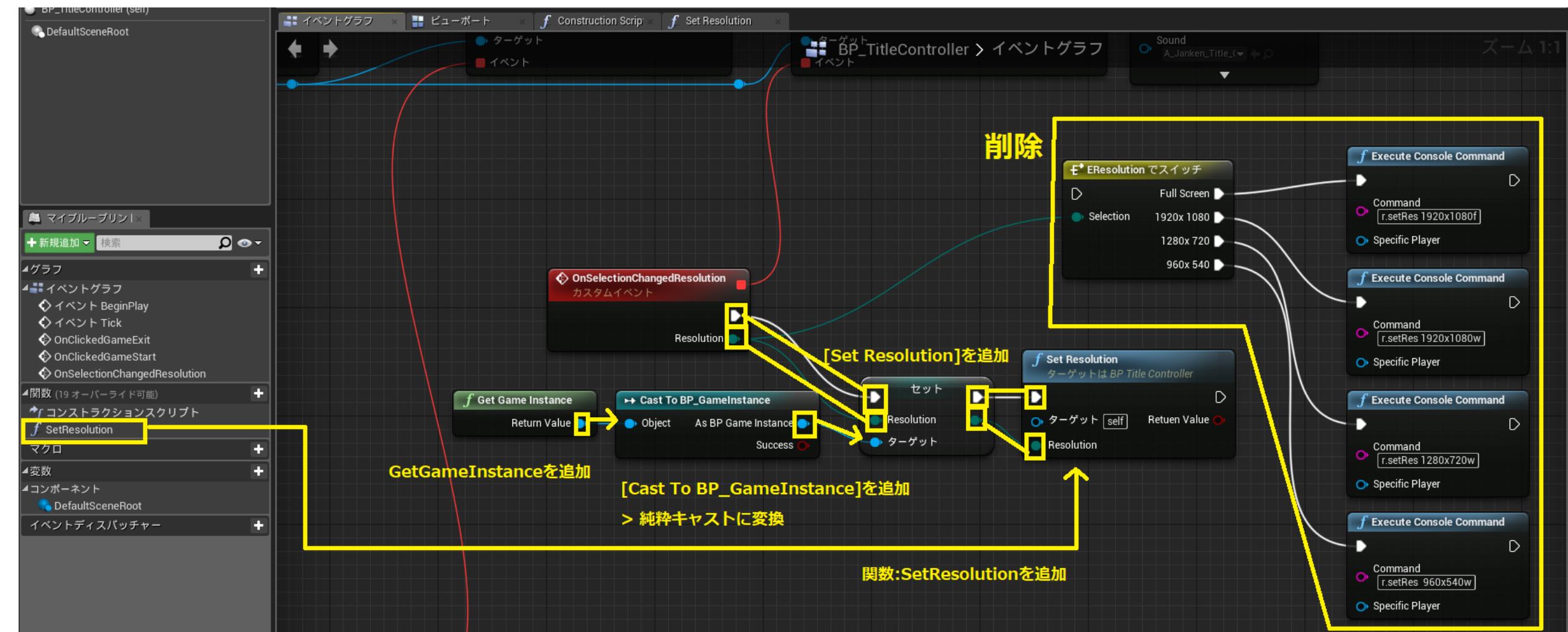
ゲームの解像度を設定する



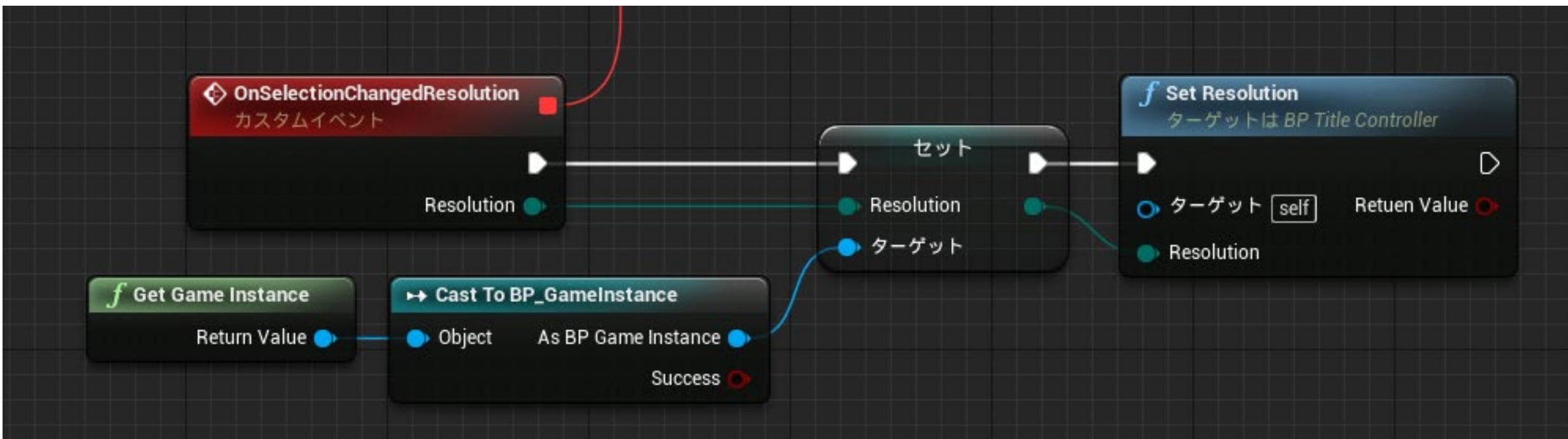
BP_TitleControllerに関するSetResolutionを追加する 2



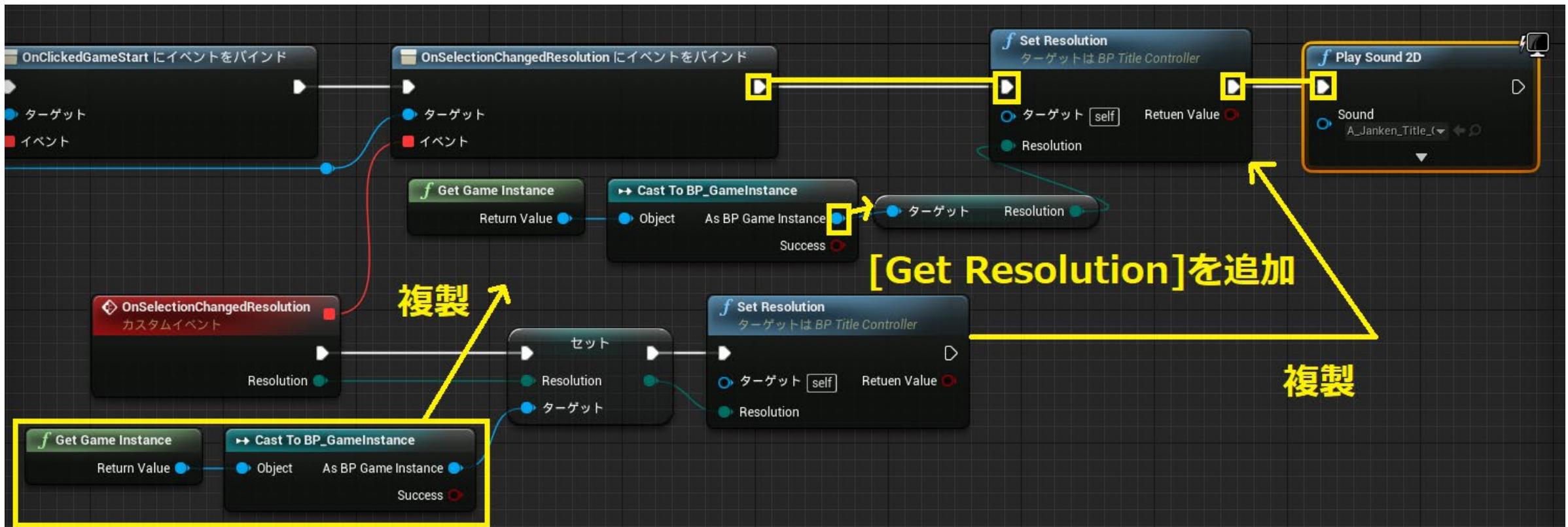
BP_TitleControllerに関するSetResolutionを追加する 3



BP_TitleControllerに関する関数:SetResolutionを追加する 4



Titleレベルを開始した時に解像度を設定

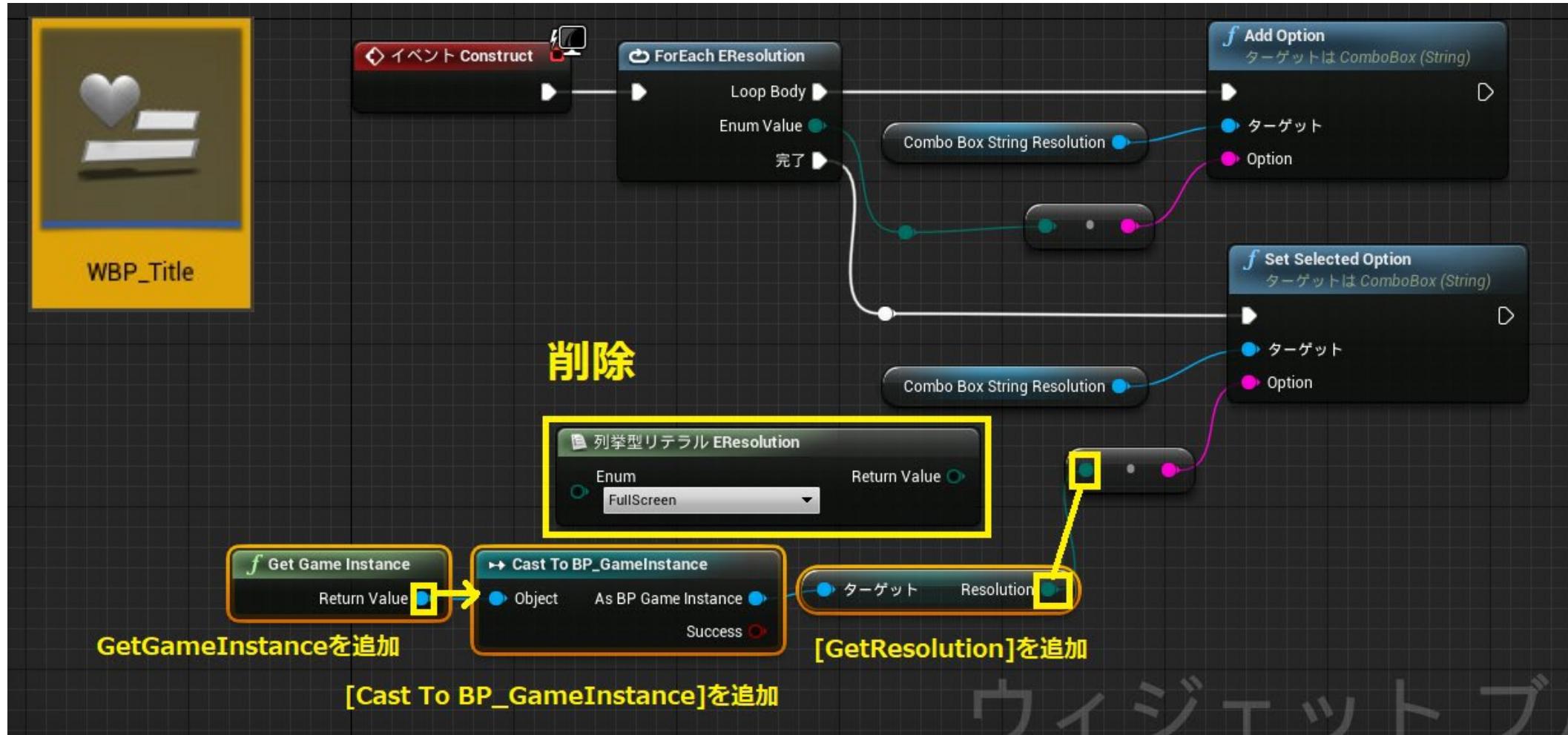


コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

WBP_TitleのイベントConstructを変更する

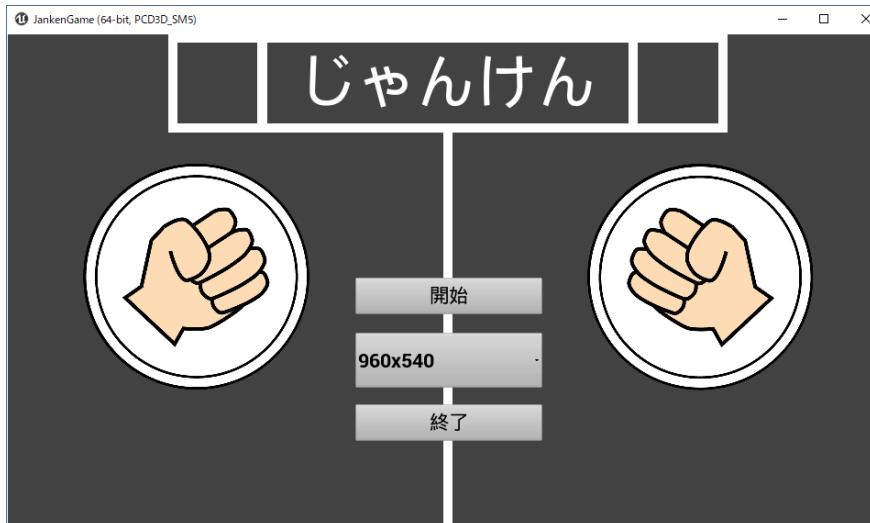


コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

プレイして確認する



解像度を960x540に変更して、開始



GameレベルからTitleレベルに遷移



960x540に設定されたまま

14. 解像度設定

14.1 列挙型 EResolutionを作成

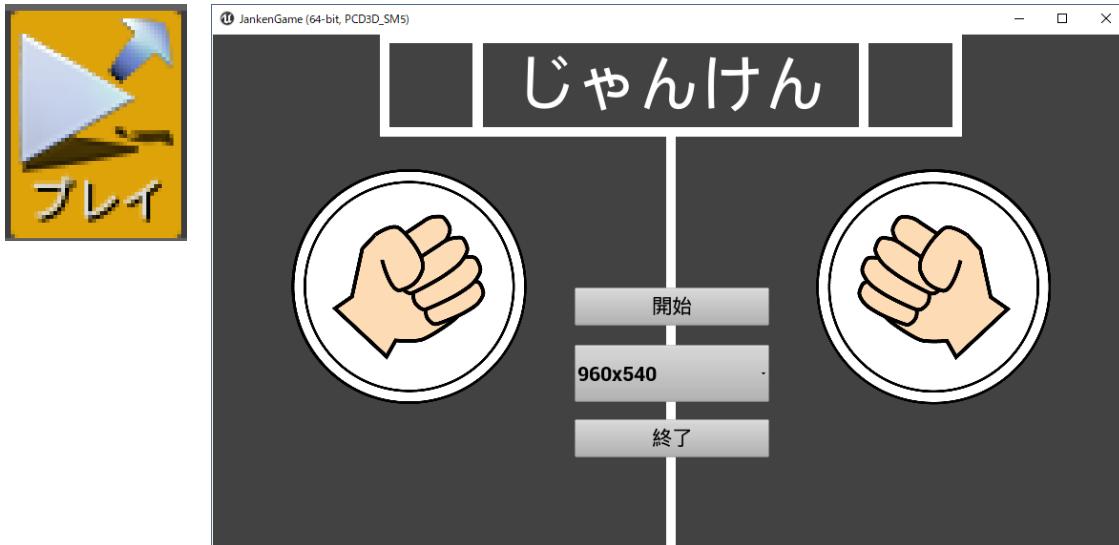
14.2 ComboBoxにEResolutionのエニュミレータをすべて表示

14.3 解像度を変える

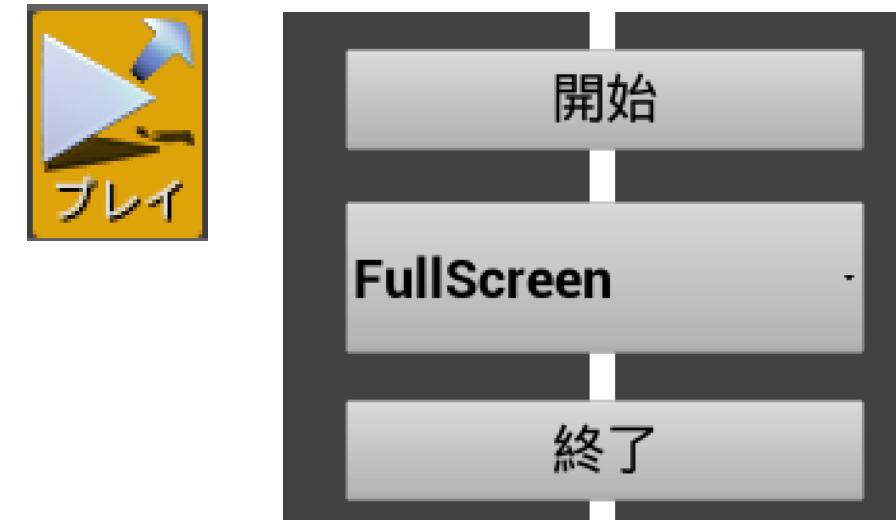
14.4 レベルを跨いで値を保持するためにBP_GameInstanceを作成

14.5 解像度の設定を保存して、ゲーム開始時に読み込む

プロジェクトを再起動すると解像度が FullScreenに戻ってしまう



解像度を960x540に変更して、終了

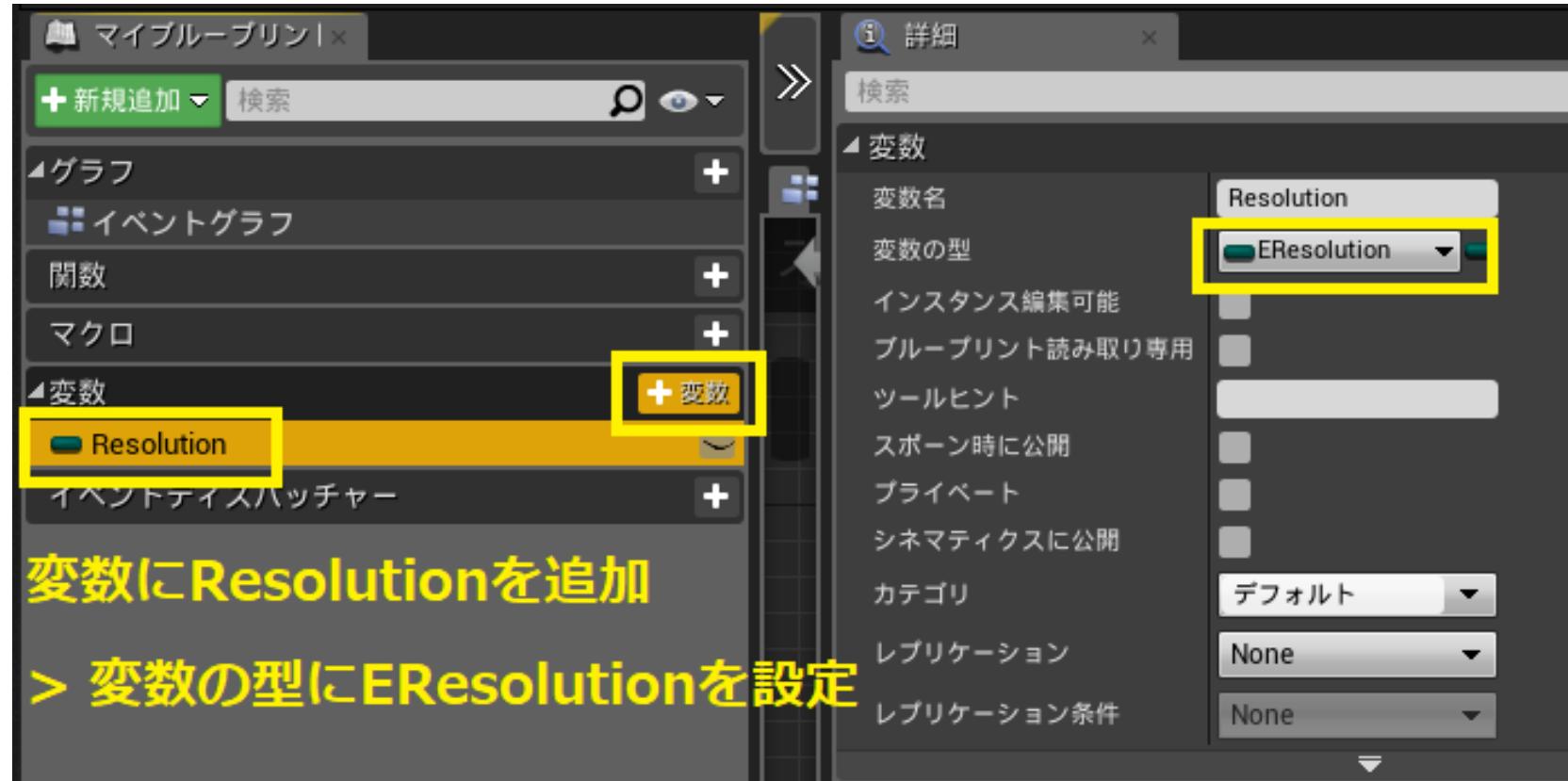


再度プレイした時に解像度がフルスクリーン
で開始される

BP_SaveGameSettings(親:SaveGame)を作成する



変数を追加する



変数名	変数の型	デフォルト値
Resolution	EResolution	- (設定なし)

コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

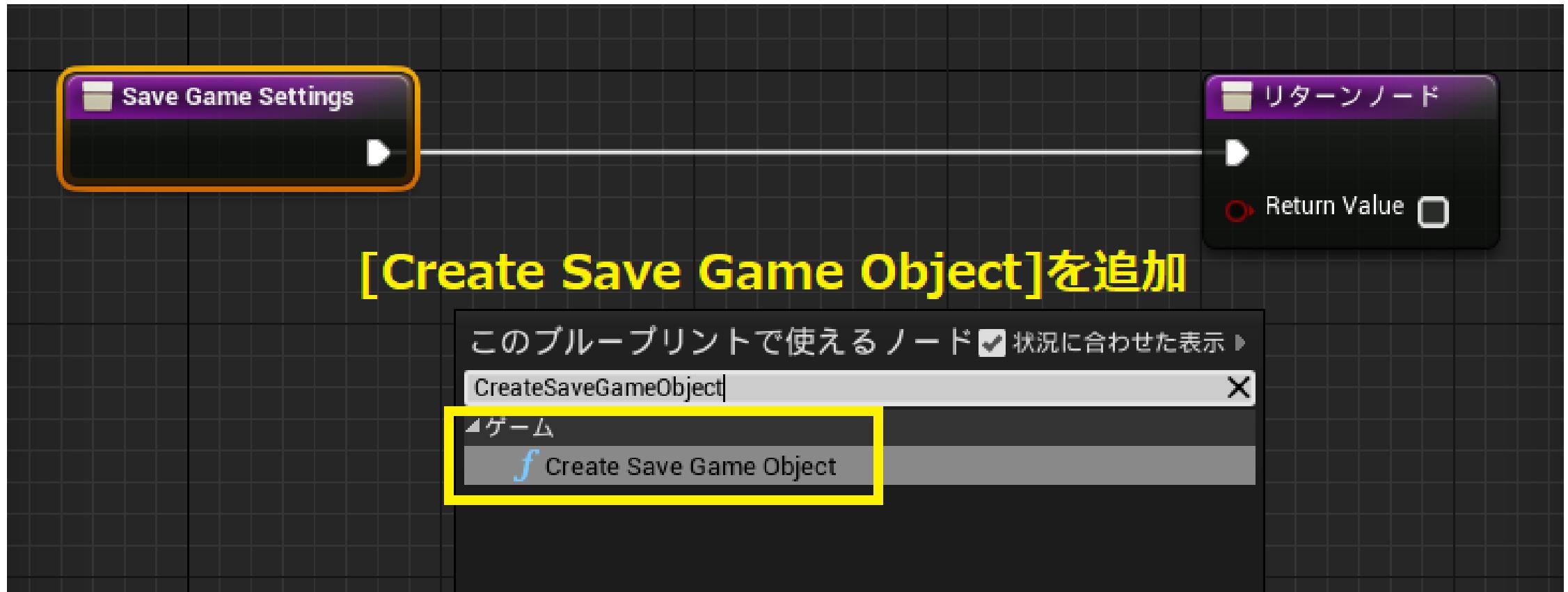
BP_GameInstanceに関する関数:SaveGameSettingsを追加する 1



関数名	アクセス指定子	Input/Output	パラメータ名	型
SaveGameSettings	パブリック	Output	ReturnValue	Boolean

説明 ゲーム設定を保存する

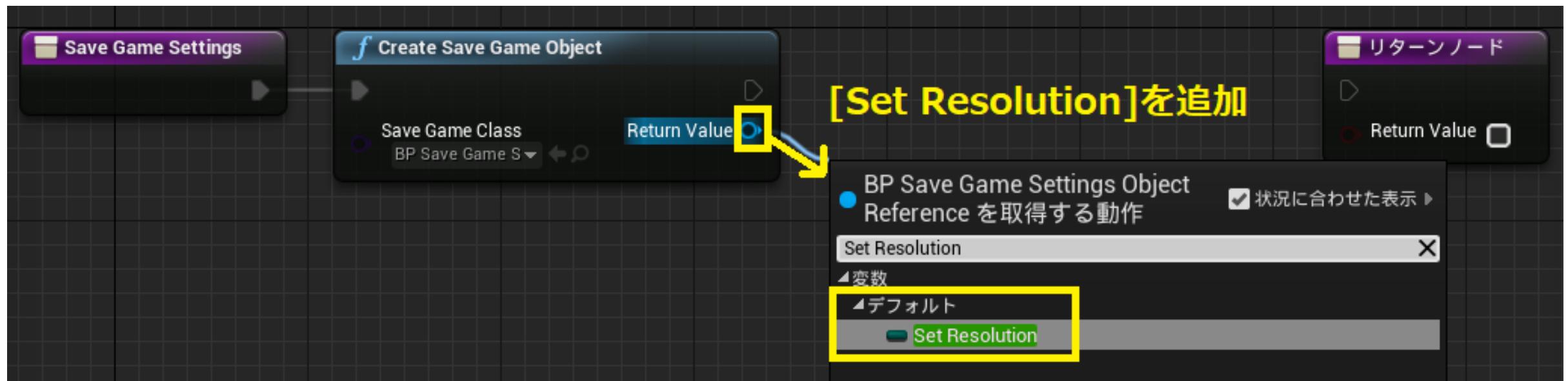
BP_GameInstanceに関するSaveGameSettingsを追加する 2



BP_GameInstanceに関数:SaveGameSettingsを追加する 3



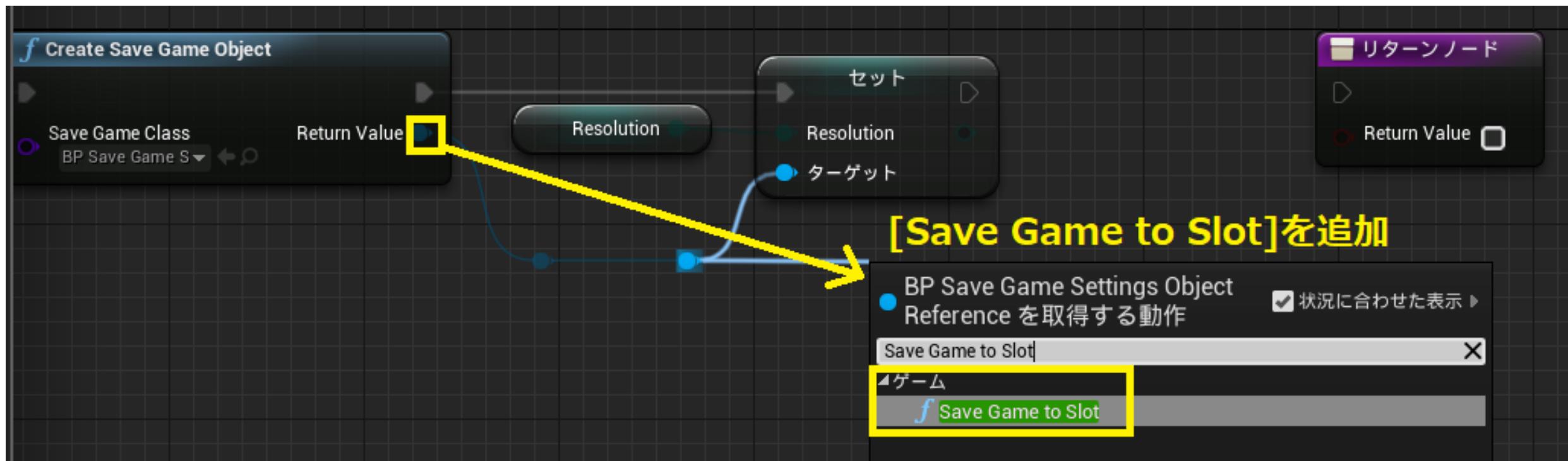
BP_GameInstanceに関するSaveGameSettingsを追加する 4



BP_GameInstanceに関するSaveGameSettingsを追加する 5



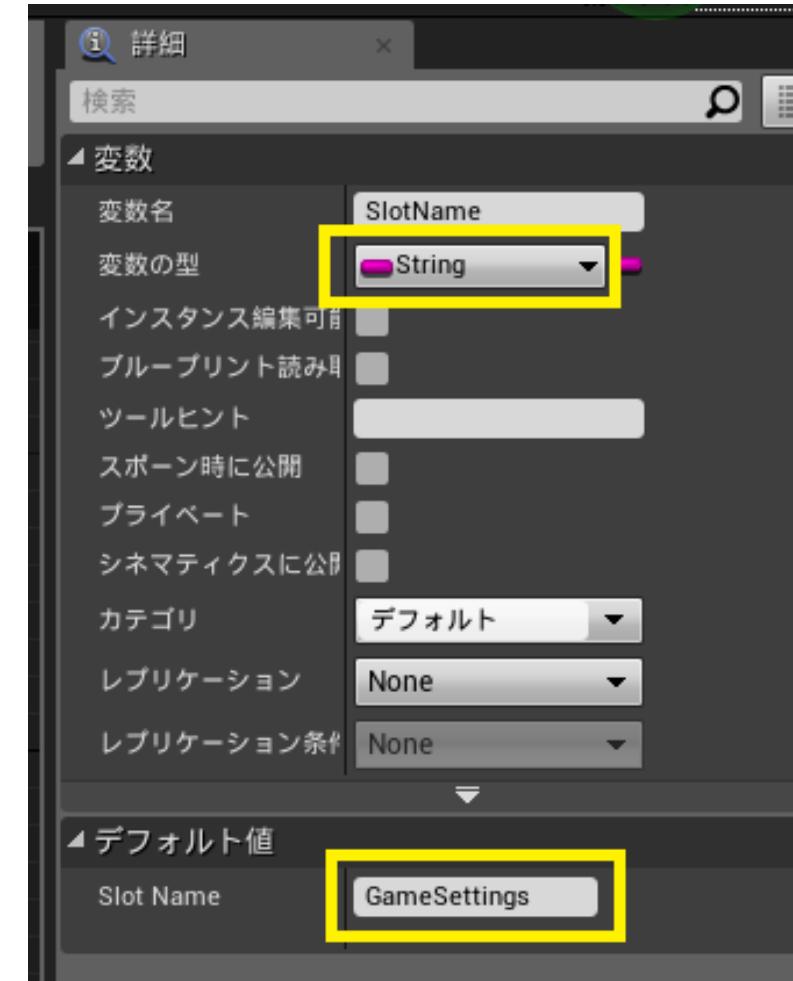
BP_GameInstanceに関するSaveGameSettingsを追加する 6



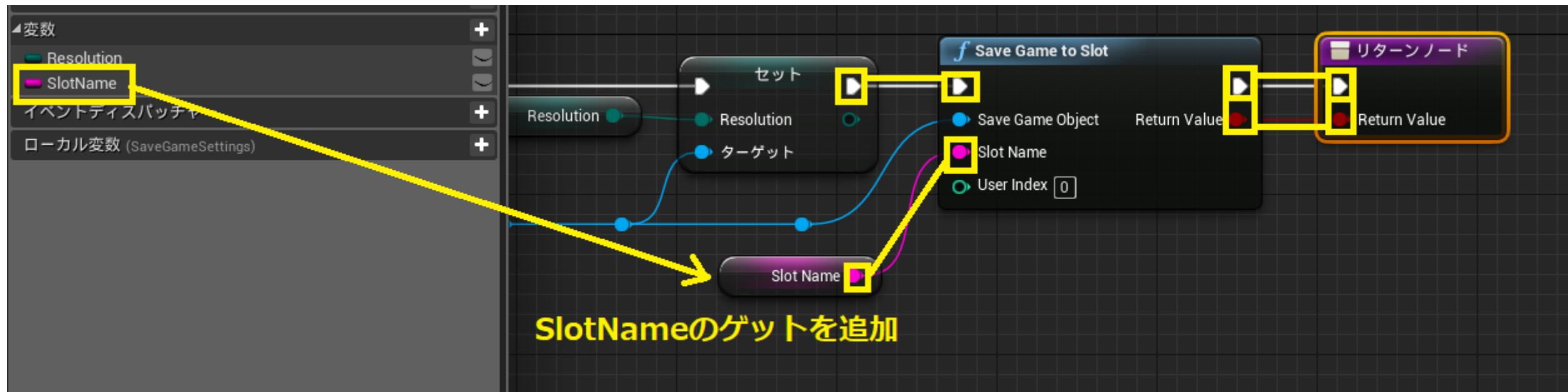
変数の追加する



変数名	変数の型	デフォルト値
Resolution	EResolution	FullScreen
SlotName	String	GameSettings



BP_GameInstanceに関数:SaveGameSettingsを追加する 7



関数:LoadGameSettingsを追加する 1

The screenshot shows the Unreal Engine Blueprint Editor interface. On the left, the main workspace displays a graph with several nodes. In the center, the 'Functions' panel lists two functions: 'SaveGameSettings' and 'LoadGameSettings'. The 'LoadGameSettings' node is highlighted with a yellow box. On the right, a detailed properties panel for 'LoadGameSettings' is open, showing its parameters: 'Description' (説明) is set to 'ゲーム設定を読み込む' (Load Game Settings), 'Category' (カテゴリ) is 'Default' (デフォルト), and 'Access Specifier' (アクセス指定子) is 'Public' (パブリック). The 'Outputs' section shows a single output named 'ReturnValue' of type 'Boolean'. Below the properties panel, a table provides a breakdown of the function's parameters:

関数名	アクセス指定子	Input/Output	パラメータ名	型
LoadGameSettings	パブリック	Output	ReturnValue	Boolean

説明 ゲーム設定を読み込む

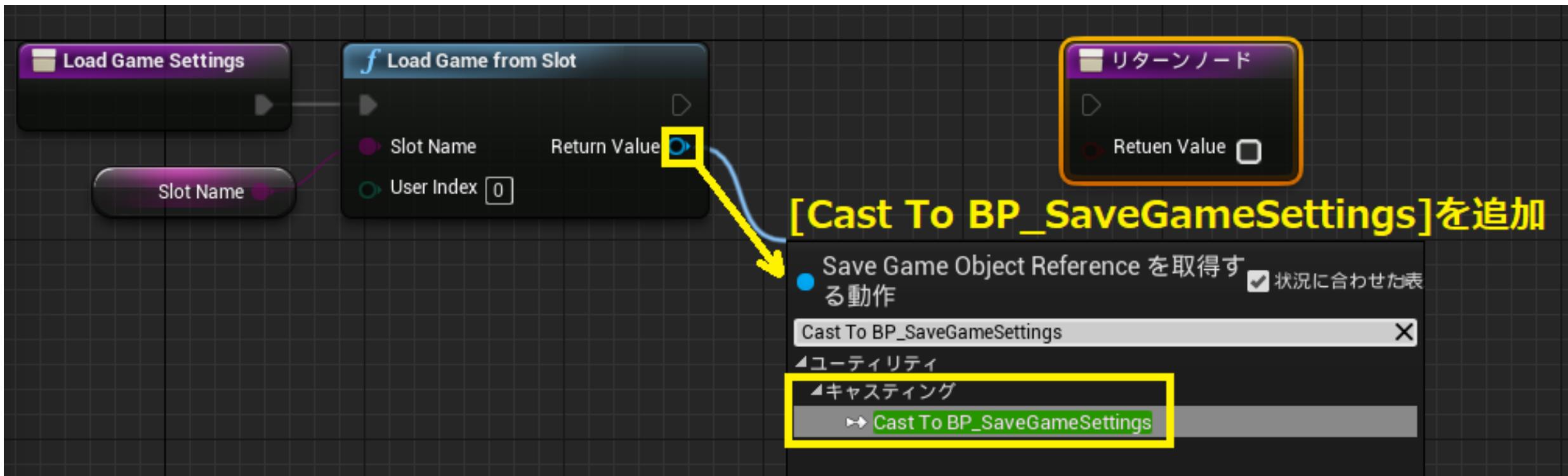
関数:LoadGameSettingsを追加する 2



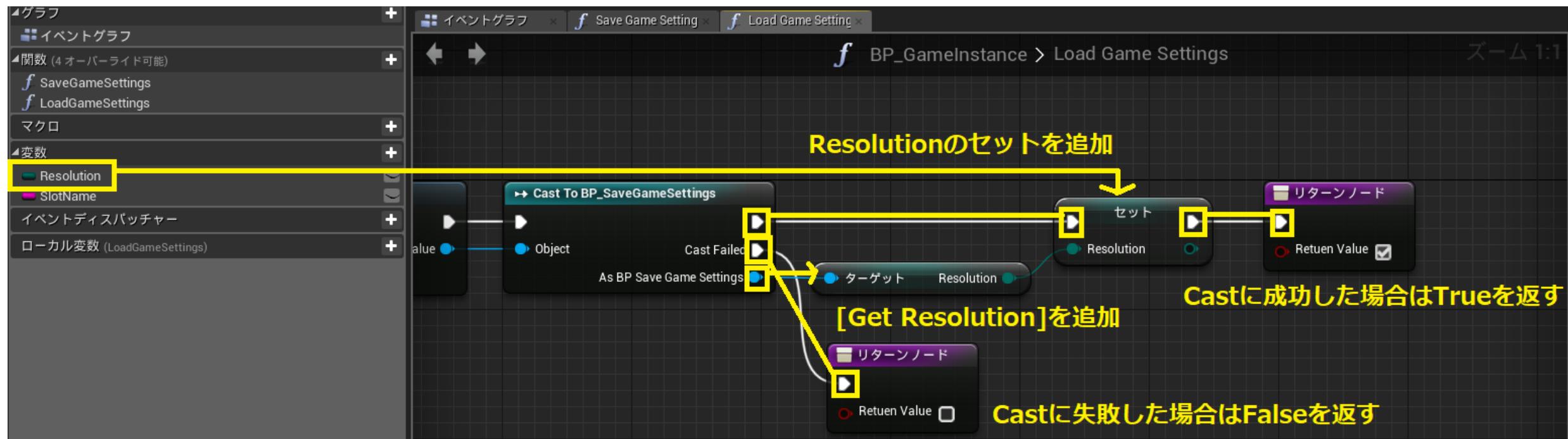
関数:LoadGameSettingsを追加する 3



関数:LoadGameSettingsを追加する 4



関数:LoadGameSettingsを追加する 5

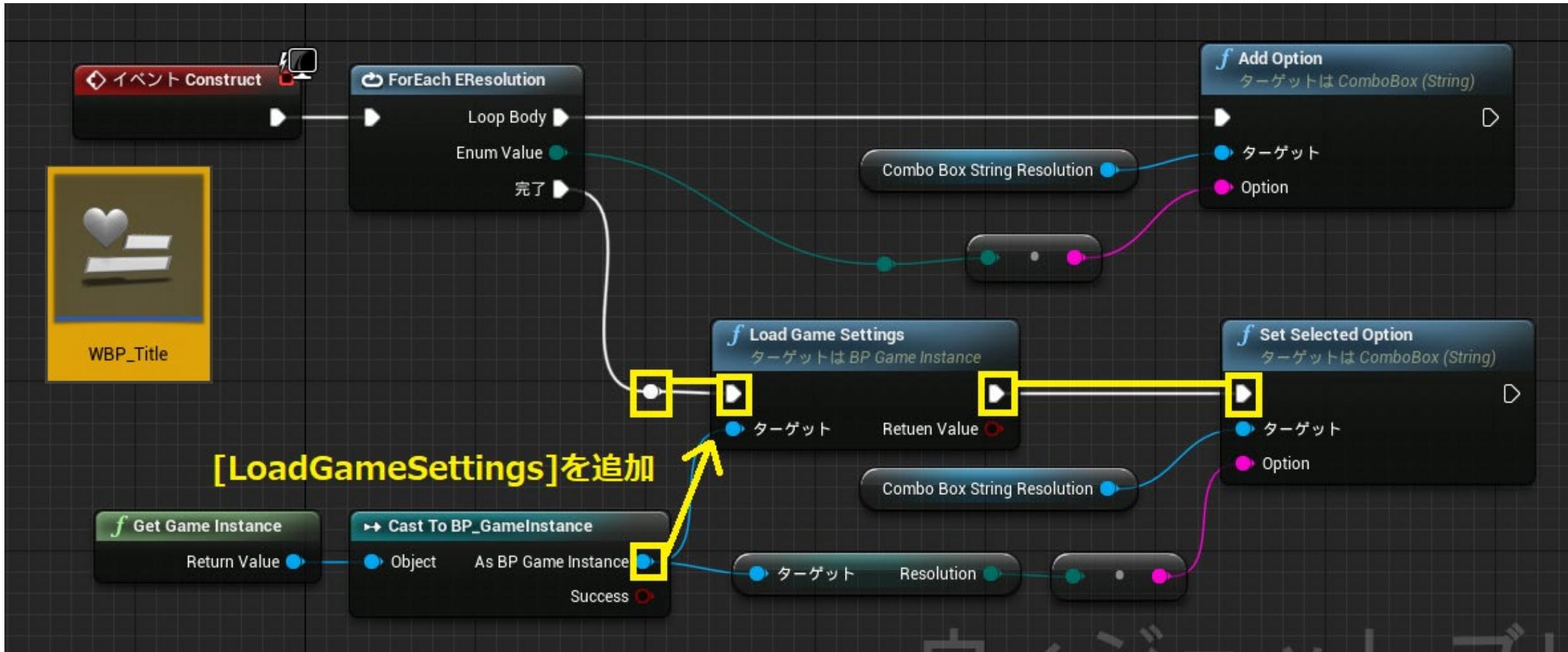


コンパイル > 保存



この後に別のブループリントを編集するため、コンパイル > 保存

WBP_TitleのイベントConstructを変更する

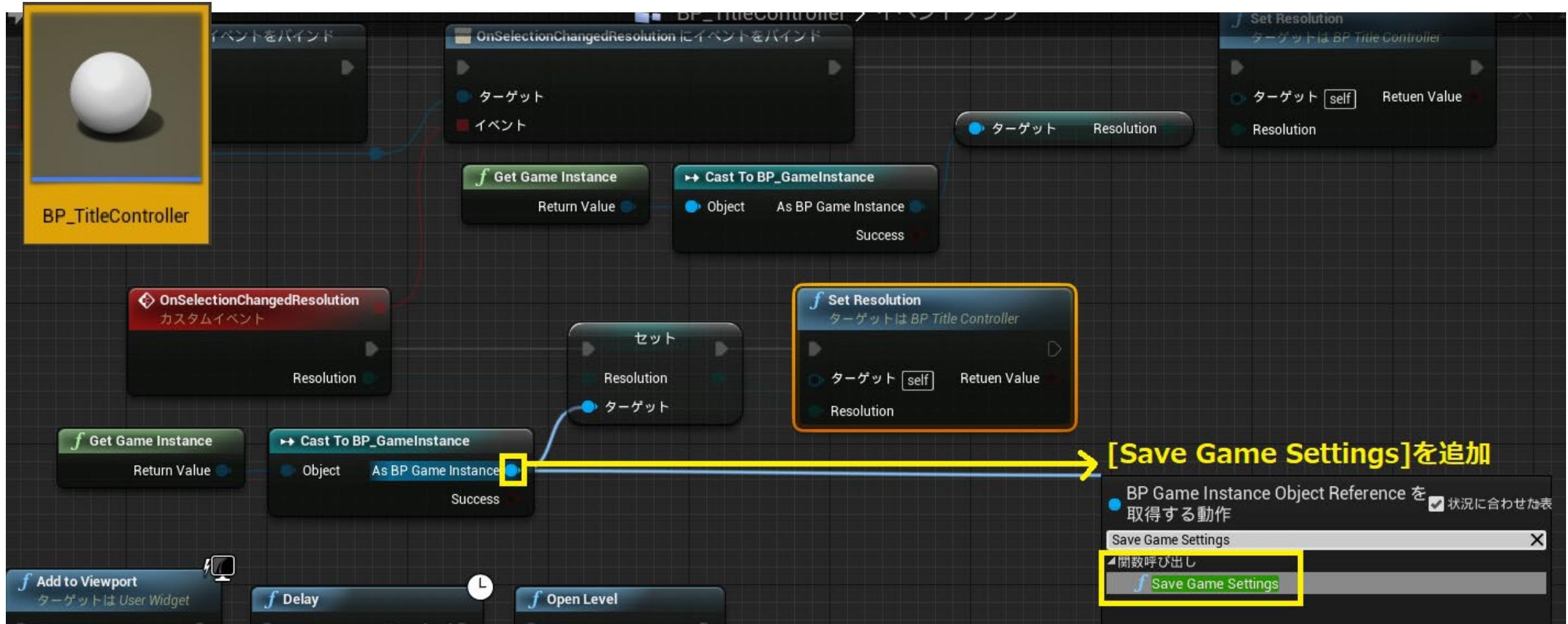


コンパイル > 保存

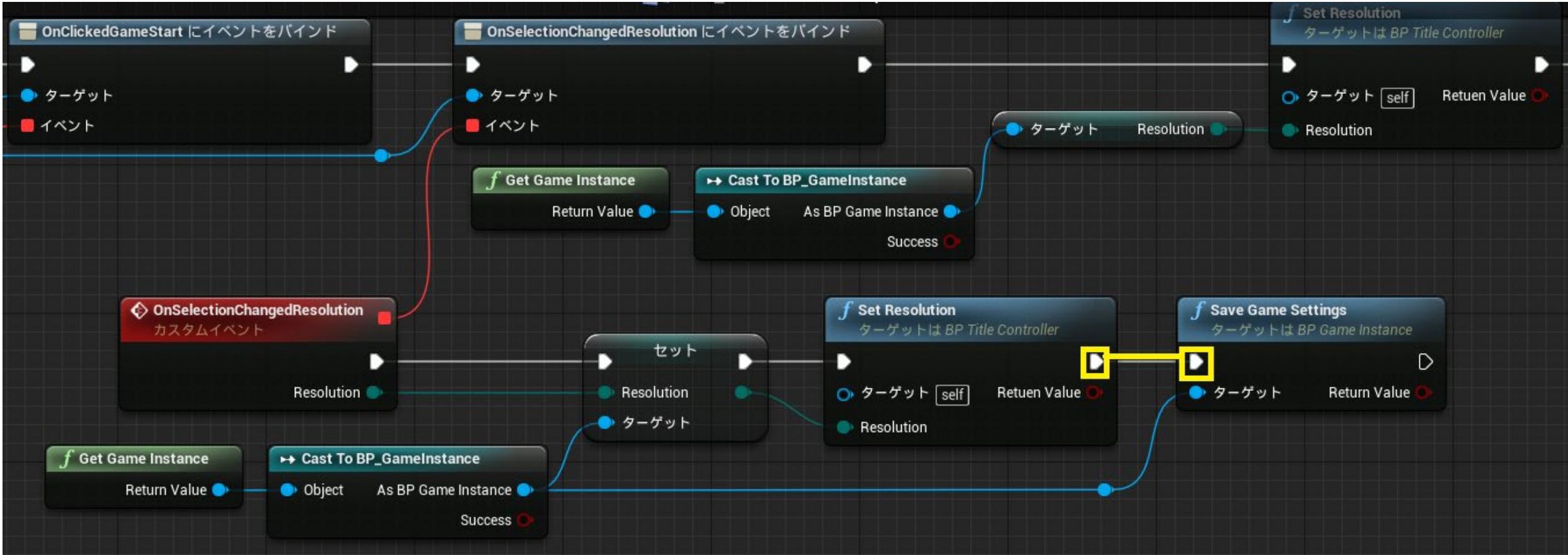


この後に別のブループリントを編集するため、コンパイル > 保存

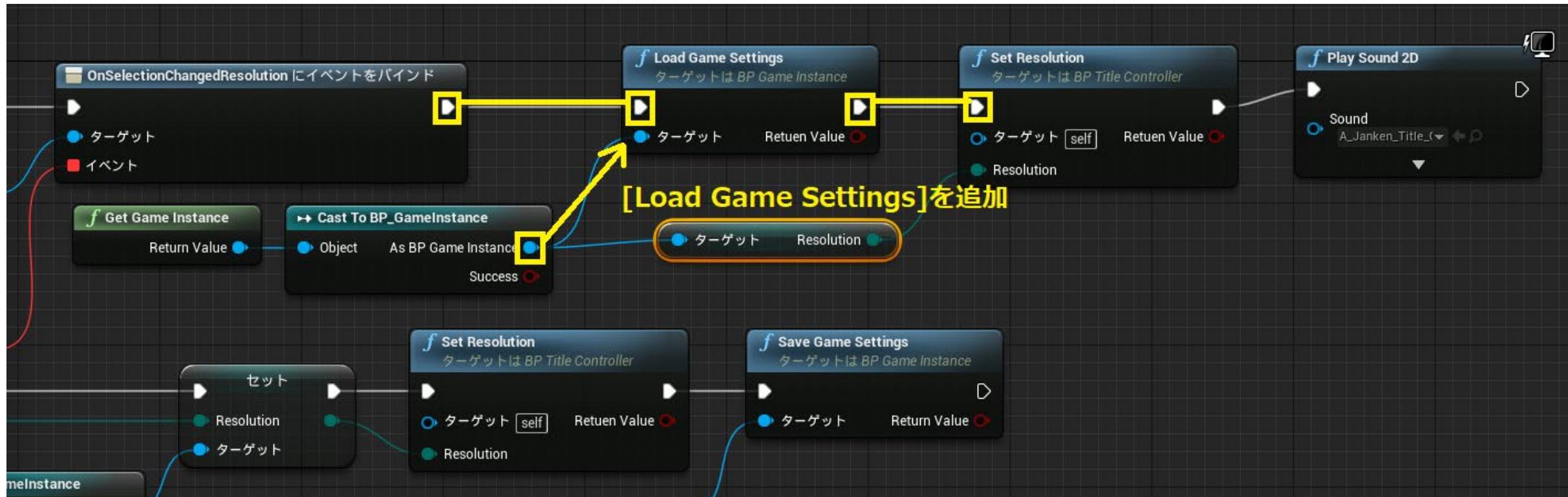
解像度を変更した際に設定を保存する 1



解像度を変更した際に設定を保存する 2



タイトル画面を開いた時に設定を読み込む

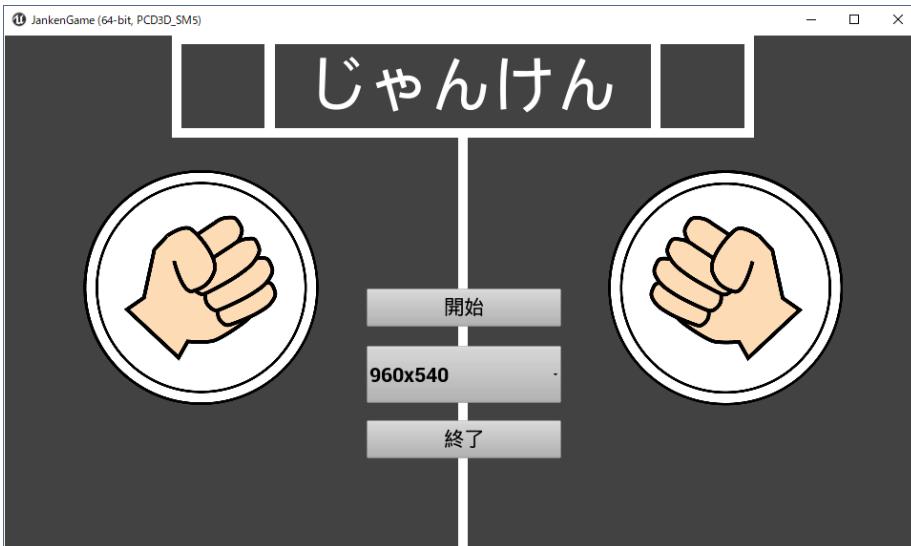


コンパイル > 保存

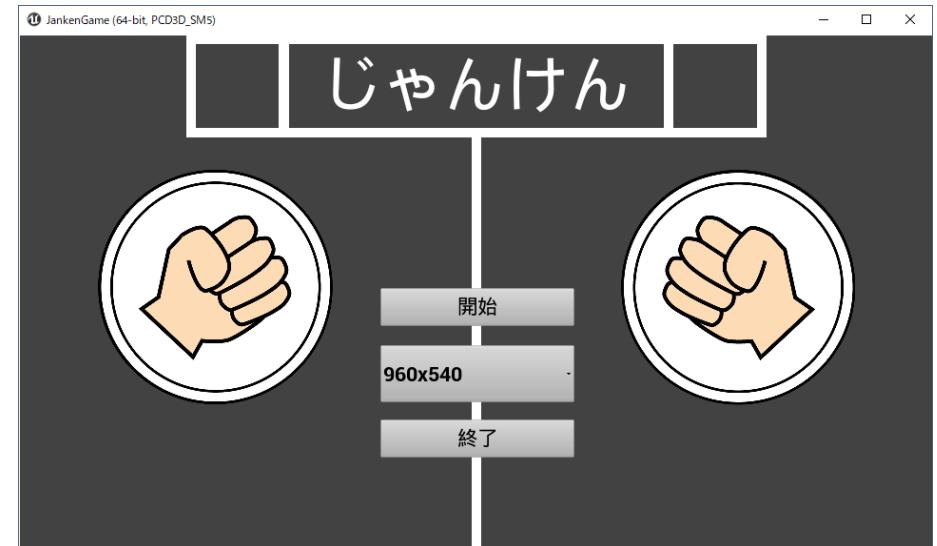


この後に別のブループリントを編集するため、コンパイル > 保存

プレイして確認する



解像度を960x540に変更して、終了



再度プレイした時に解像度が960x540で開始される

15. テスト項目の作成、デバッグ

15. テスト項目の作成、デバッグ

15.1 テスト項目票を作成して、テストを実施

じゃんけんゲームのブラックボックステストの見本ファイルをダウンロード

151. 15. テスト項目票の作成、デバッグ

152. 15.1 テスト項目票を作成して、テストを実施

見本をダウンロード

JankenGameBlackBoxTestDocument.pptx

JankenGameBlackBoxTestDocument.xlsx

PowerPoint版

項目番号	大項目	中項目	小項目	テスト内容	結果
1-2	ゲーム開始	タイトル画面の表示		タイトル画面が表示されること	
1-2-1	タイトル画面	終了ボタン	Click	ゲームが終了すること	
1-2-2		タイトルボタン	Click	ローディング画面が表示された後に、ゲーム画面が表示されること	
1-3-1	ゲーム画面	スペースバー	Press	メニュー画面が表示されること	
1-3-2		結果表示		結果画面が表示されること	
1-4-1	結果画面	再開ボタン	Click	ゲームが回目の勝負から再開されること	
1-4-2		タイトルボタン	Click	ローディング画面が表示された後に、タイトル画面が表示されること	
1-4-3		スペースバー	Press	メニュー画面が表示されること	
1-5-1	メニュー画面	再開ボタン	Click	ゲームが再開すること	
1-5-2		タイトルボタン	Click	ローディング画面が表示された後に、タイトル画面が表示されること	
1-5-3		終了ボタン	Click	ゲームが終了すること	
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					

Excel版

項目番号	大項目	中項目	小項目	テスト内容	結果
3	1-1	ゲーム開始	タイトル画面の表示	タイトル画面が表示されること	
4	1-2-1	タイトル画面	終了ボタン	ゲームが終了すること	
5	1-2-2		タイトルボタン	ローディング画面が表示された後に、ゲーム画面が表示されること	
6	1-3-1	ゲーム画面	スペースバー	メニュー画面が表示されること	
7	1-3-2		結果表示	結果画面が表示されること	
8	1-4-1	結果画面	再開ボタン	ゲームが回目の勝負から再開されること	
9	1-4-2		タイトルボタン	ローディング画面が表示された後に、タイトル画面が表示されること	
10	1-4-3		スペースバー	メニュー画面が表示されること	
11	1-5-1	メニュー画面	再開ボタン	ゲームが再開すること	
12	1-5-2		タイトルボタン	ローディング画面が表示された後に、タイトル画面が表示されること	
13	1-5-3		終了ボタン	ゲームが終了すること	
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					

練習だと思って、テンプレートから作成してもらつてもらうのが理想です

▶ 51.8. テスト項目票の作成、デバッグ

▶ 52.8.1 そのゲームちゃんと動くの？

▶ 53.8.2 ゲーム内容からテスト項目の抽出

▶ 54.8.3 テスト項目票を作成して、テストを実施

 [MathGameBlackBoxTestDocument.pptx](#)

 [MathGameBlackBoxTestDocument.xlsx](#)

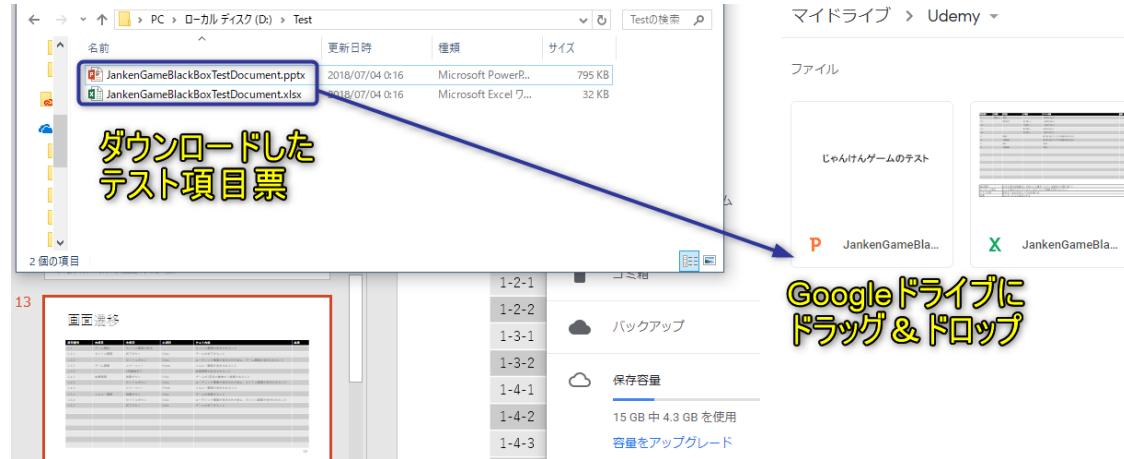
**算数ゲーム
8章を復習して作成出来るのが理想**

▶ 55.8.4 ブループリント、関数単位のテストするためのデバッグ方法について

▶ 56.8.5 ブラックボックステスト、ホワイトボックステスト

▶ 57.8.6 作りたいゲームの作り方、工程に対応するテスト

ExcelやPowerPointを持っていない人は Google ドライブにアップすることで編集可能



Google スpreadsheetで開く

	A	B	C	D	テスト内容
1	項目番号	大項目	中項目	小項目	
2	1-1	問題出力	数値A		?が表示される
3	1-2-1		数学記号	足し算(+)	+が表示される
4	1-2-2			引き算(-)	-が表示される
5	1-2-3			掛け算(*)	*が表示される
6	1-2-4			割り算(/)	/が表示される
7	1-3		数値B		繰り返し毎にランダムの数値が表示される
8	1-4		計算結果		繰り返し毎にランダムの数値が表示される
9	1-5		解答		非表示
10	1-6		結果画像		非表示
11					
12					

Google スpreadsheetで開く

PCにExcelが入っていないなくても
Googleスプレッドシートで編集できる

	A	B	C	F	
1	画面遷移				
2	項目番号	大項目	中項目	小項目	テスト内容
3	1-1	ゲーム開始	タイトル画面の表示		タイトル画面が表示されること
4	1-2-1	タイトル画面	終了ボタン	Click	ゲームが終了すること
5	1-2-2		タイトルボタン	Click	ローディング画面が表示された後に、ゲーム画面が表示されること
6	1-3-1	ゲーム画面	スペースバー	Press	メニュー画面が表示されること
7	1-3-2		5回勝負完了		結果画面が表示されること
8	1-4-1	結果画面	再開ボタン	Click	ゲームが1回目の勝負から再開されること
9	1-4-2		タイトルボタン	Click	ローディング画面が表示された後に、タイトル画面が表示されること
10	1-4-3		スペースバー	Press	メニュー画面が表示されること
11	1-5-1	メニュー画面	再開ボタン	Click	ゲームが再開すること
12	1-5-2		タイトルボタン	Click	ローディング画面が表示された後に、タイトル画面が表示されること
13	1-5-3		終了ボタン	Click	ゲームが終了すること
14					

16. パッケージ化

16. パッケージ化

16.1 配布用パッケージにするためのプロジェクト設定

16.2 Windows(64ビット)でパッケージ化

16.3 パッケージサイズをより小さくする

16. パッケージ化

16.1 配布用パッケージにするためのプロジェクト設定

16.2 Windows(64ビット)でパッケージ化

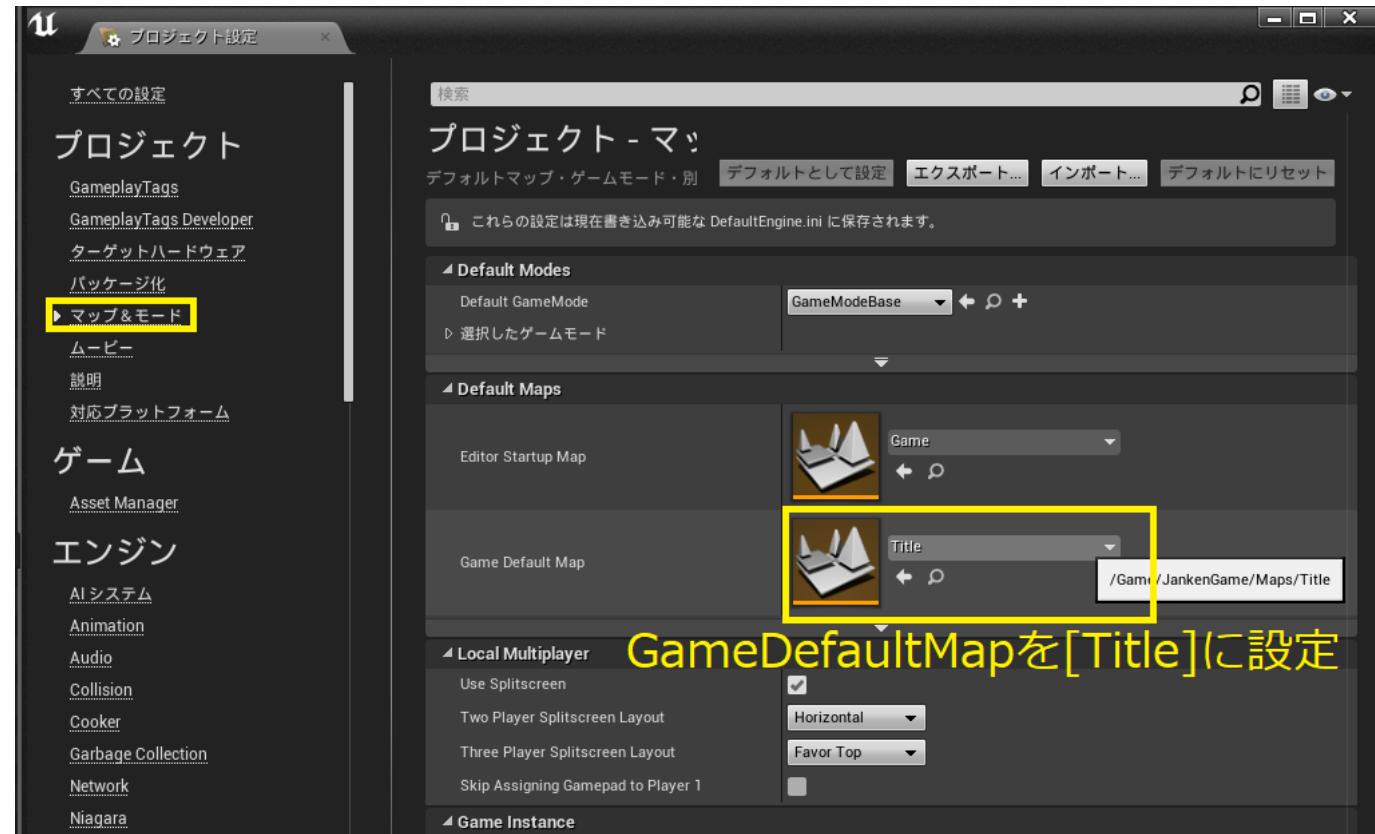
16.3 パッケージサイズをより小さくする

プロジェクト設定を開く



設定 > プロジェクト設定

ゲームを開始した時に1番最初に開くレベルを設定する

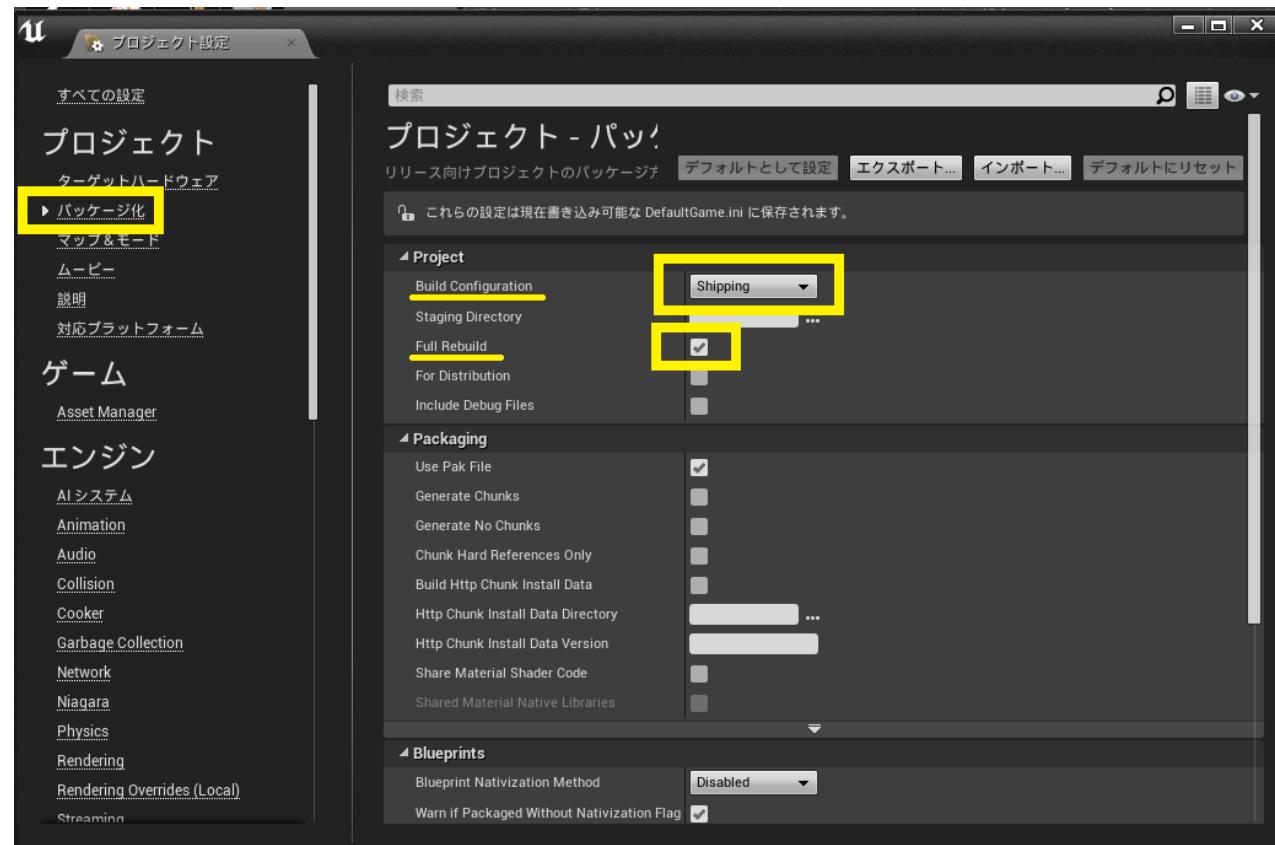


プロジェクト | マップ & モード

項目	設定
GameDefaultMap	Title

Game Default Mapは
ゲームを開始した時に1番最初に開くレベルを設定する

パッケージを出荷用のビルド設定にする



プロジェクト | パッケージ化を選択
Build Configuration, Full Rebuild を変更する

プロジェクト | パッケージ化

項目	設定
Build Configuration	Shipping
Full Rebuild	チェック

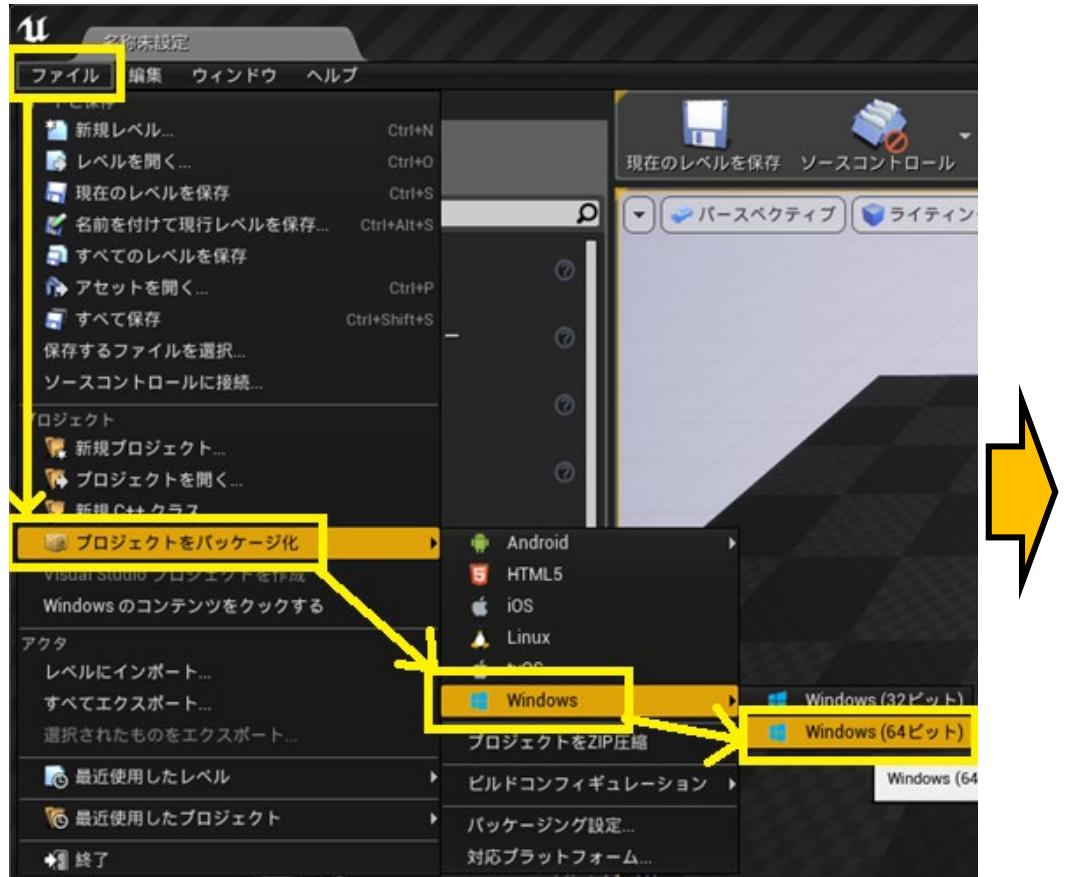
16. パッケージ化

16.1 配布用パッケージにするためのプロジェクト設定

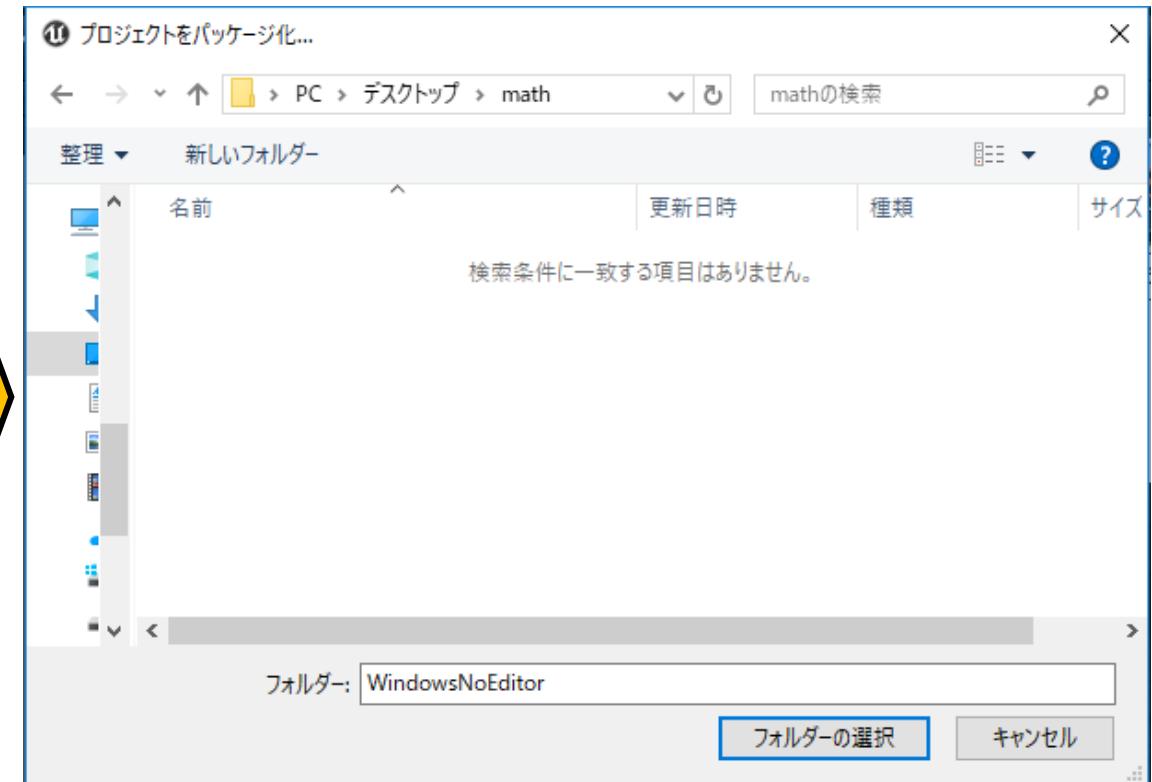
16.2 Windows(64ビット)でパッケージ化

16.3 パッケージサイズをより小さくする

プロジェクトをパッケージ化 (Windows(64ビット))

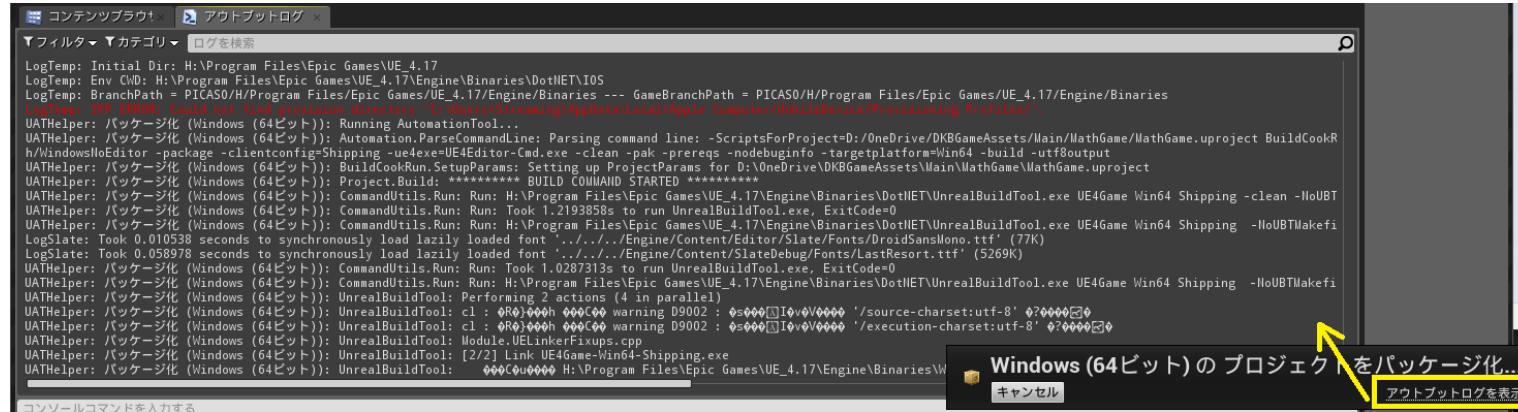


ファイル > プロジェクトをパッケージ化
> Windows > Windows(64ビット)



パッケージを書き出すフォルダを選択
> フォルダの選択

アウトプットログウィンドウでビルド状況を確認

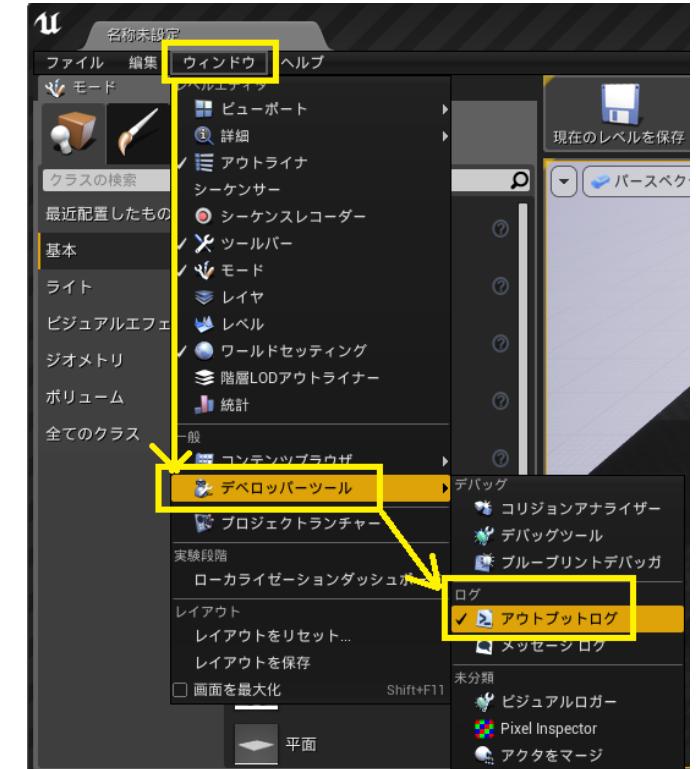


右下にダイアログが表示される
アウトプットログを表示をクリックすると
アウトプットログウィンドウが表示される



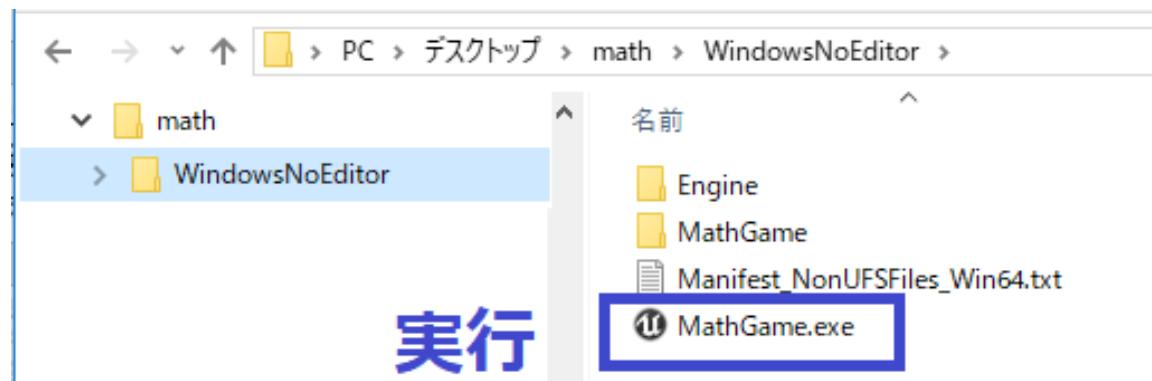
```
UATHelper: パッケージ化 (Windows (64ビット)): UnrealPak: LogPakFile: Display: Added 1399 files, 112359431 bytes total, time 3.06s.  
UATHelper: パッケージ化 (Windows (64ビット)): UnrealPak: LogPakFile: Display: Unreal pak executed in 3.107124 seconds  
UATHelper: パッケージ化 (Windows (64ビット)): CommandUtils.Run: Run: Took 3.5506962s to run UnrealPak.exe, ExitCode=0  
UATHelper: パッケージ化 (Windows (64ビット)): Project.RunUnrealPak: UnrealPak Done *****  
UATHelper: パッケージ化 (Windows (64ビット)): Project.CopyManifestFilesToStageDir: Copying NonUFSFiles to staging directory: D:\OneDrive  
UATHelper: パッケージ化 (Windows (64ビット)): Project.CopyBuildToStagingDirectory: ***** STAGE COMMAND COMPLETED *****  
UATHelper: パッケージ化 (Windows (64ビット)): Project.Package: ***** PACKAGE COMMAND STARTED *****  
UATHelper: パッケージ化 (Windows (64ビット)): Project.Package: ***** PACKAGE COMMAND COMPLETED *****  
UATHelper: パッケージ化 (Windows (64ビット)): Project.Archive: ***** ARCHIVE COMMAND STARTED *****  
UATHelper: パッケージ化 (Windows (64ビット)): Project.Archive: ***** ARCHIVE COMMAND COMPLETED *****  
UATHelper: パッケージ化 (Windows (64ビット)): Automation.Execute: BUILD SUCCESSFUL  
UATHelper: パッケージ化 (Windows (64ビット)): Program.Main: AutomationTool exiting with ExitCode=0 (Success)
```

BUILD SUCCESSFULが表示されればパッケージ化成功

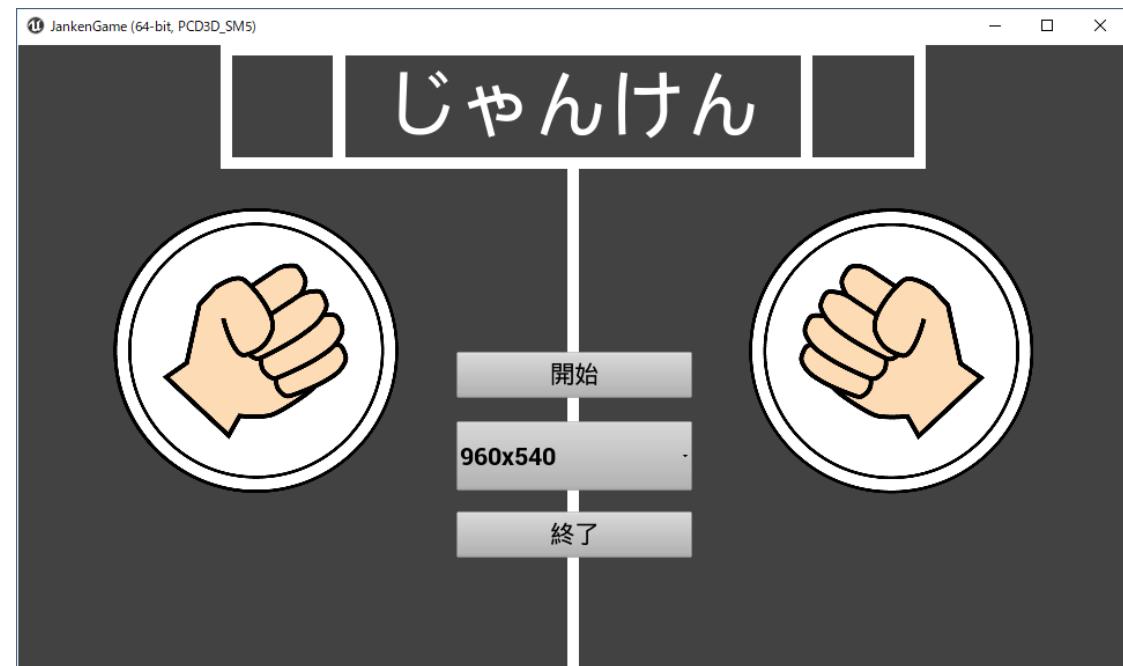


アウトプットログの表示手順
ウインドウ > デベロッパーツール
> アутプットログ

選択したフォルダにWindowsNoEditorフォルダが作成され、実行ファイルが出力される



選択したフォルダにWindowsNoEditorフォルダが作成される
JankenGame.exeを実行する



ゲームが開始される
WindowsNoEditorフォルダ配下を全部コピーすれば、
他のPCでも実行することが出来る

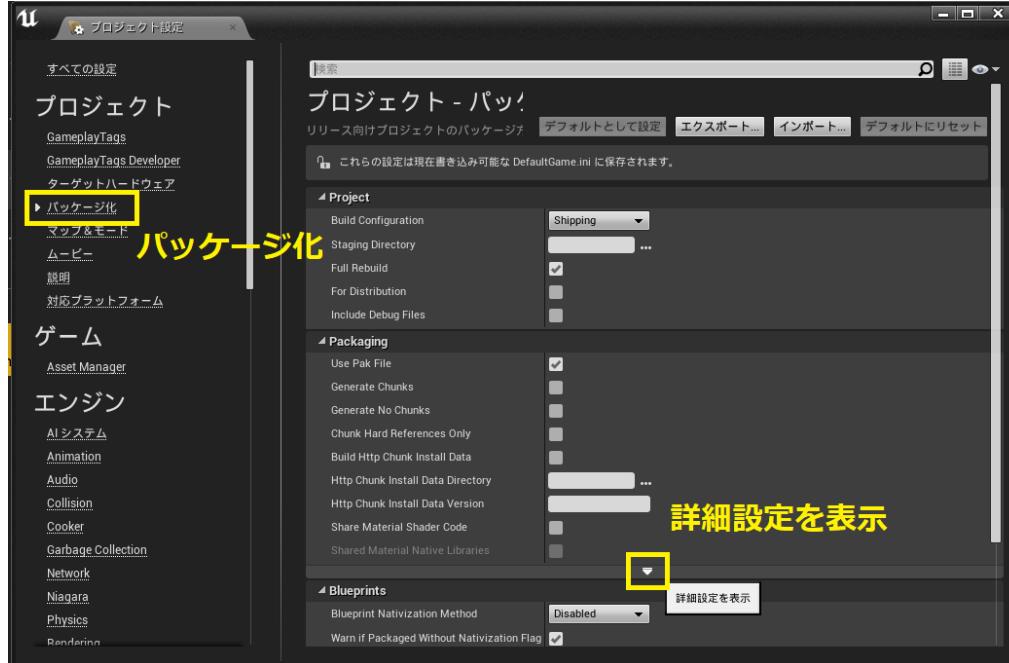
16. パッケージ化

16.1 配布用パッケージにするためのプロジェクト設定

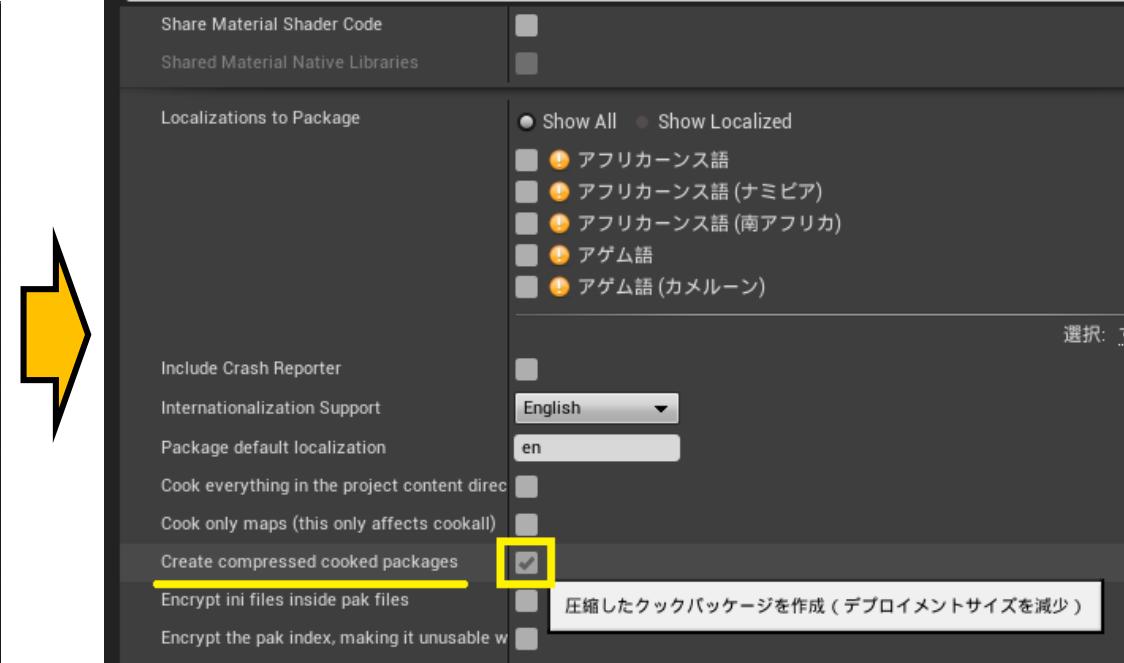
16.2 Windows(64ビット)でパッケージ化

16.3 パッケージサイズをより小さくする

クックしたパッケージを圧縮する（1番小さくなる） Create compressed cooked packagesにチェック



パッケージ化 > Packaging カテゴリの▼をクリック



Create compressed cooked packageにチェック

合計210 MB > 138 MBまで減った

パッケージ化したゲームのサイズを小さくする方法（他にも色々ある方法があるが、自己責任で）

<https://docs.unrealengine.com/latest/JPN/Engine/Performance/ReducingPackageSize/index.html>