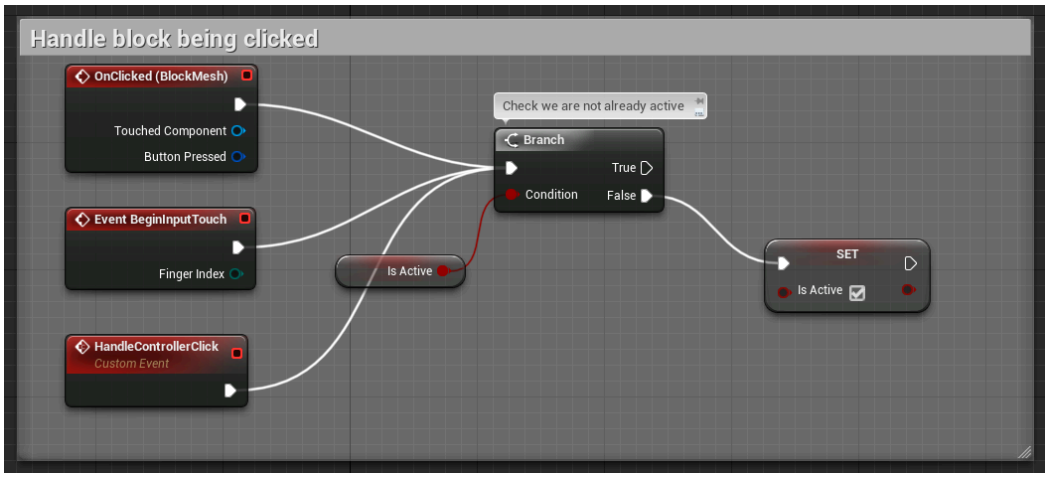
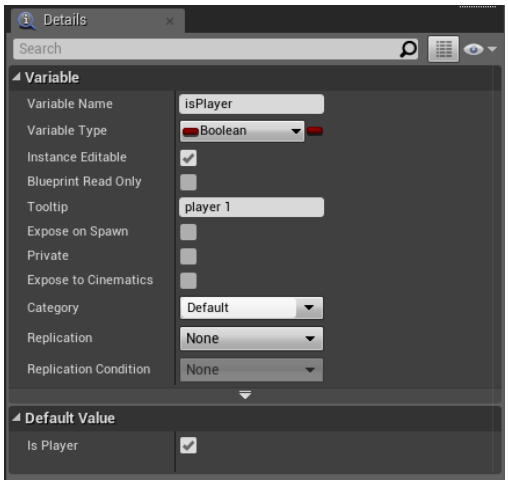
TRES EN RAIA

Creamos un proxecto novo de tipo puzle.
Encontraremos dous Blueprints principais cos que vamos a traballar: PuzzleBlock e PuzzleBlockGrid.
Podemos borrar parte da programación; no BP Puzzle Block (PB)eliminamos o cambio de material e a puntuación da sección *Handle block being clicked*.



No BP PuzzleBlockGrid (PBGrid) encontramos a creación do taboleiro ao inicio do xogo e unha sección de puntuación, *Handle scoring when block is clicked*, que podemos eliminar, xunto coa variable *Score* e a compoñente *ScoreText*.

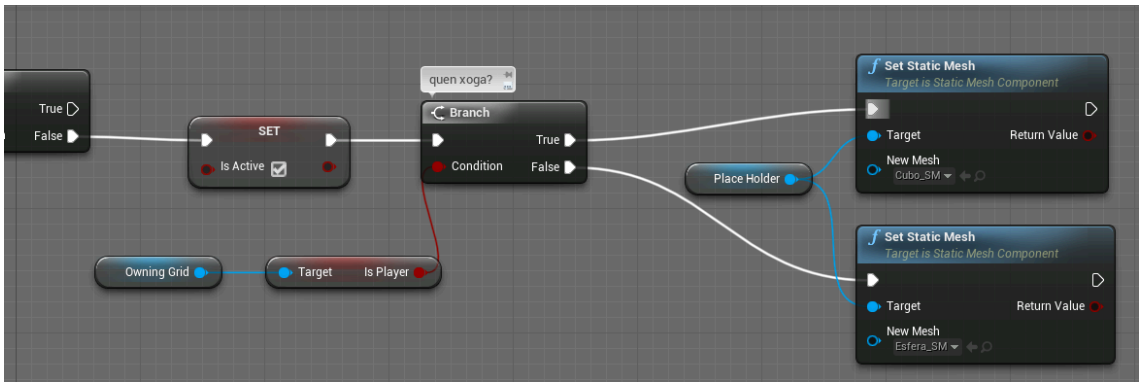
O primeiro que temos que facer é crear dous símbolos para xogar, poden ser un cubo e unha esfera de diferentes cores que converteremos en *Static Mesh*.
Durante o xogo teremos que saber a que xogador lle toca xogar, para iso creamos unha variable booleana no PBGrid á que chamamos *isPlayer* e asignámoslle o valor 1 (Player 1 será o valor 1 e Player 2 o valor 0).



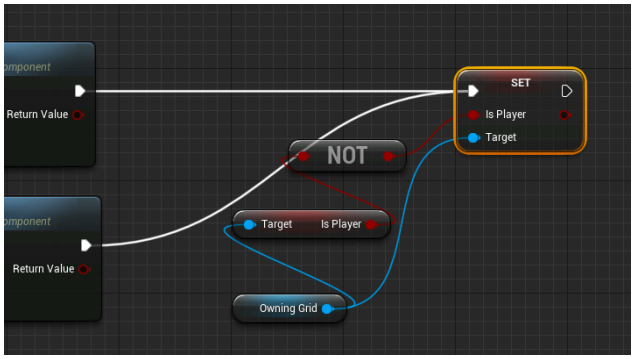
Para comprobar que player está xogando empregaremos unha comunicación directa con Blueprint. No PB temos unha variable chamada *Owning Grid*. É unha referencia de

obxecto ao noso taboleiro desde a que podemos acceder a calquera variable ou función almacenada en PBGrid e que empregaremos para acceder á variable *isPlayer*.

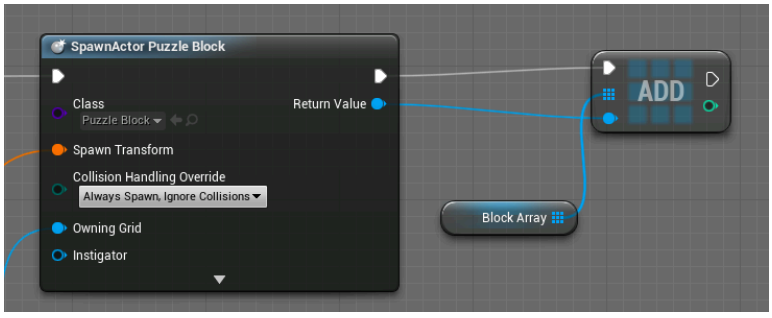
Antes de nada crearemos un soporte para as fichas no taboleiro, para iso, no Viewport do PB agregaremos unha compoñente StaticMesh á que chamaremos *PlaceHolder* (0,0,70). Agora, no EventGraph, colocaremos ao final da sección *Handle block being clicked* un cubo se xoga o Player1 e unha esfera cando é o turno de Player2.



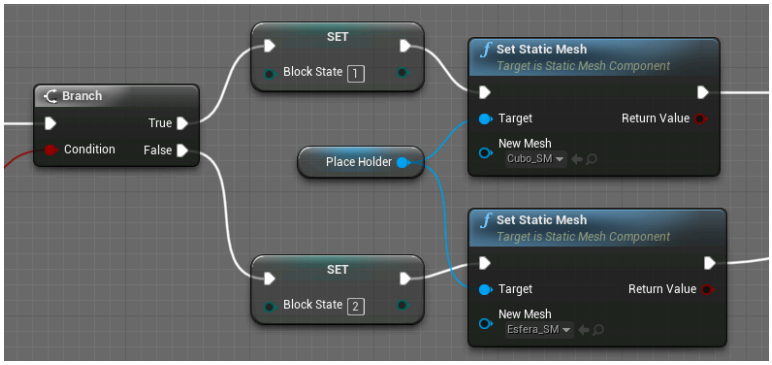
Se probamos o xogo neste momento veremos que se coloca sempre a mesma peza, porque aínda non implementamos o cambio de quenda. Para iso, despois de colocar unha peza cambiamos o valor da variable *isPlayer*.



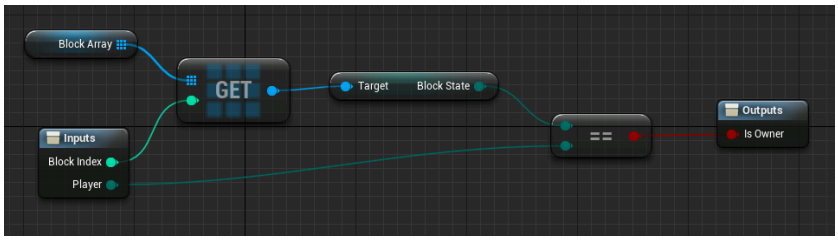
Para poder comprobar se un xogador gaña necesitamos un array de todos os bloques e o seu estado. No PBGrid crearemos unha variable array de tipo *PuzzleBlock* á que chamaremos *BlockArray*. Despois de crear cada un dos bloques, engadímolos ao *BlockArray.*



Agora, necesitamos almacenar una variable que indique o estado do bloque. Empregaremos unha variable tipo byte: *BlockState* no PB. Un bloque pode ter tres estados: 0 = vacío, 1 = player1 e 2 = player2. Empregaremos esta variable cando revisemos o array para identificar rapidamente a que xogador pertence ese bloque. Asignamos o valor desta variable cando colocamos os elementos sobre os bloques.

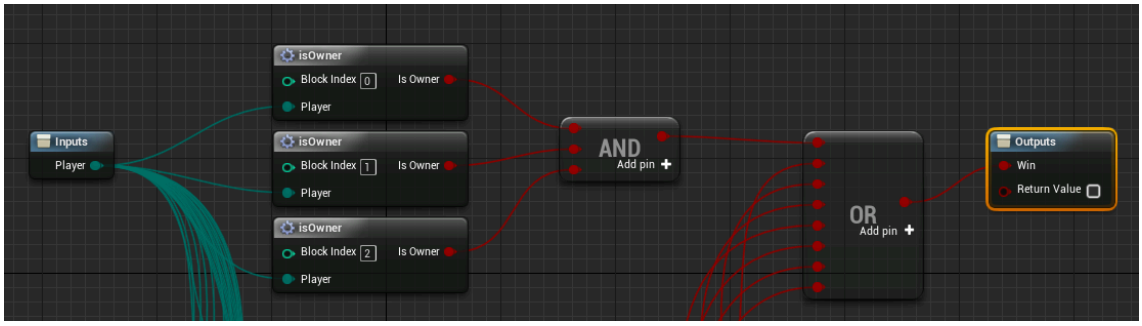


Para encontrar un gañador necesitamos comprobar en todas as combinacións posibles (3 filas + 3 columnas + 2 diagonais) se a combinación seleccionada ten o mesmo estado.
Para facer estas comprobacións crearemos dúas macros. A primeira comparará o estado do bloque cunha variable e a segunda compara todas as combinacións posibles para detectar unha combinación gañadora.
No PBGrid creamos unha nova macro á que chamaremos *isOwner*, que devolverá o valor *true* se o xogador é o propietario do bloque analizado. Definimos para esta macro dúas entradas, un valor de tipo enteiro chamado *BlockIndex* e outro de tipo byte ao que chamaremos *Player*. A saída será unha variable booleana chamada *IsOwner*.

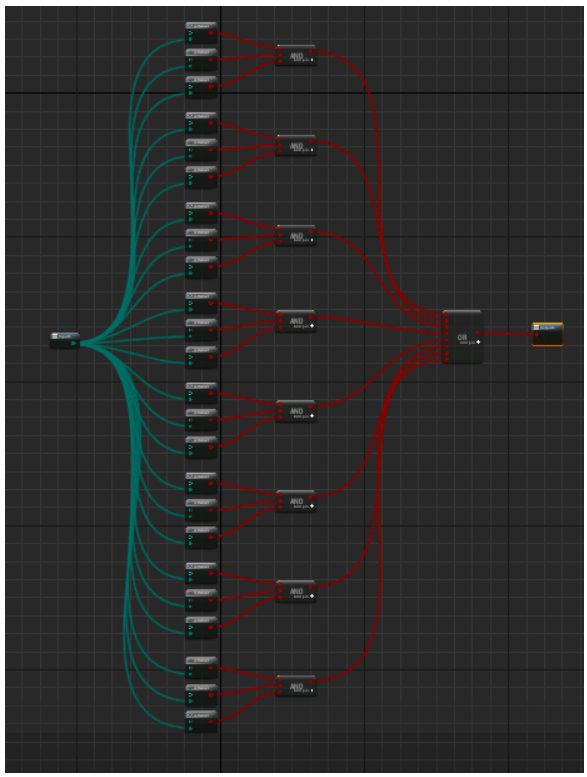


Á segunda macro chamarémola *CheckIfWin*. Neste caso só precisamos una entrada de tipo byte chamada *Player* e unha saída de tipo booleano chamada *Win*. Este nodo devolverá *true* se o xogador é propietario de todos os bloques que compoñen calquera das oito combinacións gañadoras, noutro caso devolverá *false*.
Esta é táboa de índices da matriz que representa o noso taboleiro:

2	5	8
1	4	7
0	3	6



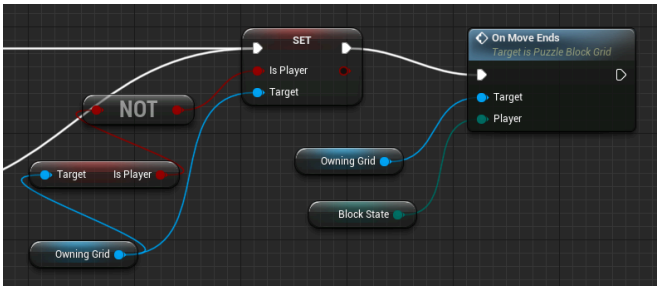
O resultado é unha grande cantidade de fíos e nodos repetitivos (que poderían ser mellorados empregando funcións e outras optimizacións):



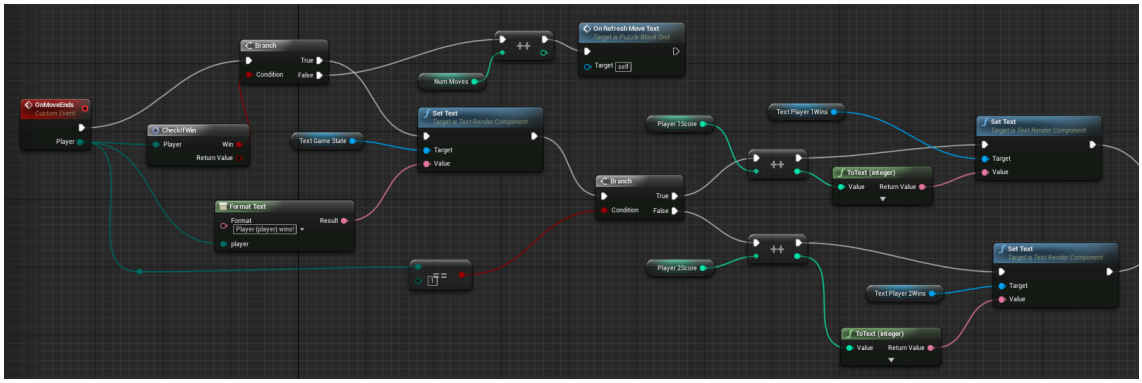
Agora só falta mostrar os resultados, farémolo con compoñentes *TextRender* que manexamos desde o PBGrid. A nosa interface de usuario contará cun estado de xogo, dous contadores de partidas gañadas e dous textos estáticos (un para cada player); así, engadimos 5 compoñentes *TextRender*:

- TextGameState: (230, -150, 0), 90º Y, tamaño 48 [Game State]
- TextPlayer1Tooltip: (-600, 450, 0) 90º Y, tamaño 72 [Player1], blue
- TextPlayer2Tooltip: (-600, -900, 0) 90º Y, tamaño 72 [Player2], red
- TextPlayer1Wins: (-400, 400, 0) 90º Y, tamaño 144 [0], blue
- TextPlayer2Wins: (-400, -950, 0) 90º Y, tamaño 144 [0], red

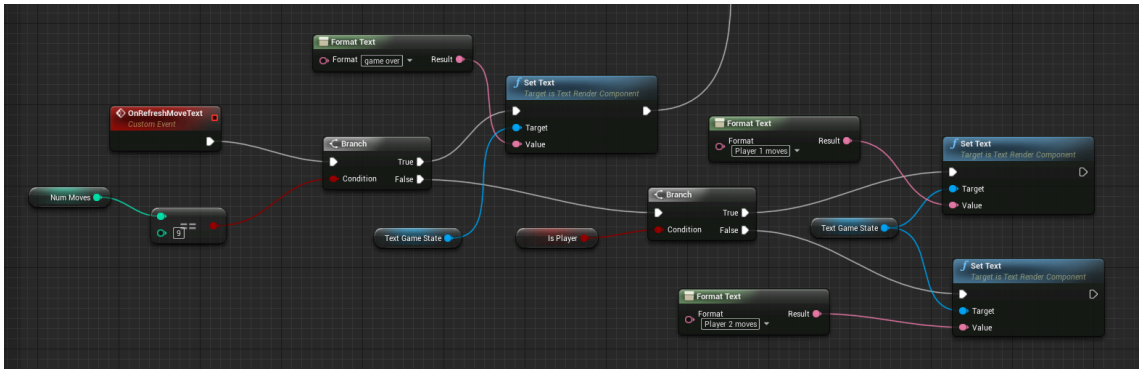
Para actualizar a interface crearemos un evento personalizado ao que chamamos *OnMoveEnds*, que terá un *input* de tipo Byte chamado *Player*.



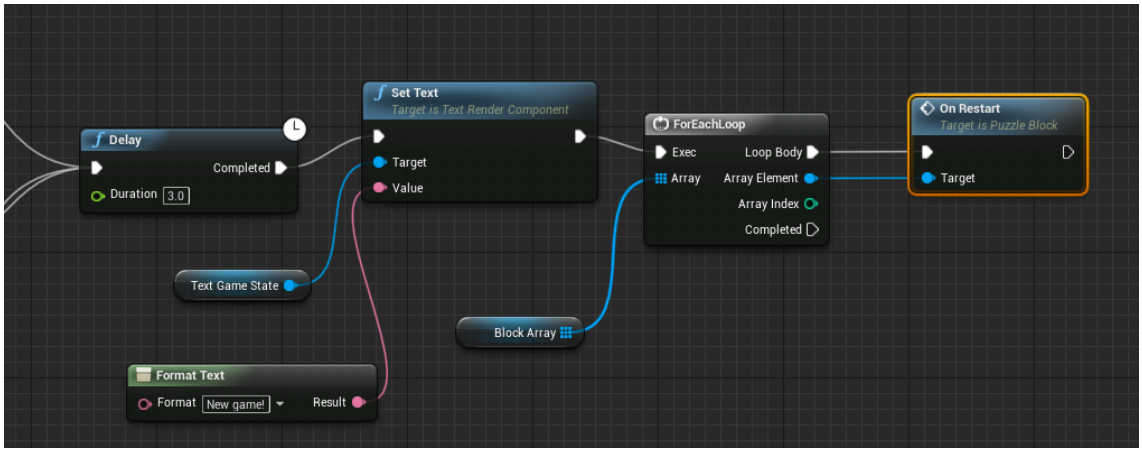
Despois de cada movemento lanzaremos este evento (despois de facer o cambio de quenda), onde comprobamos se o xogador gañou a partida, se é así indicámolo, sumamos 1 ao contador do xogador e rematamos o xogo. En caso contrario indicamos que é a quenda do outro player. Se xa non hai máis movemento posibles rematamos o xogo. Despois de rematar o xogo agardamos uns segundos e comezamos outro novo. Será necesario crear 3 novas variables de tipo enteiro: *Player1Score*, *Player2Score* e *NumMoves*, todas con valor inicial 0.



Para mostrar o cambio de quenda empregaremos outro evento personalizado ao que chamamos *OnRefreshMoveText.* Se non acadamos o número máximo de movementos cambiamos a quenda e mostramos a que Player lle toca xogar. Chamaremos a este evento tamén ao iniciar o xogo, unha vez xerado o taboleiro (como último nodo despois de BeginPlay), para que os xogadores saiban quen comeza.



Por último, teremos que crear un método no PB que reinicie o xogo. Co nodo *ForLoop*, chamamos ao método *OnRestart* para cada un dos elementos que contén o array. Non debemos esquecer colocar a variable NumMoves a 0 (falta na imaxe).



Para cada elemento do array poñeremos IsActive a *false*, a variable BlockState a 0, eliminamos as fichas e cambiamos o material do taboleiro.