언리얼 패스트 트랙
워크샵 3

워크샵 3: HUD, 생명력 시스템, 그리고 고득점 기능 만들기

학습 포인트

테스트 드라이브(60분)

- UMG 소개
- 언리얼 엔진에서 UI와 HUD 만들기
- 게임에 메뉴 시스템과 일시 정지 기능 추가하기

그랑프리(90분)

- 복잡한 HUD 만들기
- 생명력, 대미지, 그리고 점수 시스템 만들기
- 레벨 완료 시스템 만들기

오프로딩 및 토론(90분)

- UI 다시 만들기
- 게임에 독특한 메커니즘 추가하기

테스트 드라이브

3주 차에서는 UMG, '언리얼 모션 그래픽'을 중점적으로 다룹니다. UI 요소 제작에 있어서 UMG 시스템의 원리를 이해하는 건 매우 중요합니다. 이번 강좌는 'Your First Hour with UMG' 입니다. 강좌는 https://www.unrealengine.com/en-US/onlinelearning-courses/your-first-hour-with-umg 에서 확인할 수 있습니다.

이 강좌에 사용되는 프로젝트를 다운로드하려면 마켓플레이스에서 코드를 사용해야 합니다. www.unrealengine.com 에서 로그인하고, 우측 상단의 닉네임 위에 마우스를 올립니다. **개인**을 클릭합니다. 계정 페이지가 열리고, 왼쪽에 옵션 목록이 표시됩니다. 목록 아래에 '**코드 사용**'이 있습니다. 강좌에 있는 코드를 붙여넣으면 라이브러리에서 프로젝트를 잠금 해제할 수 있습니다.

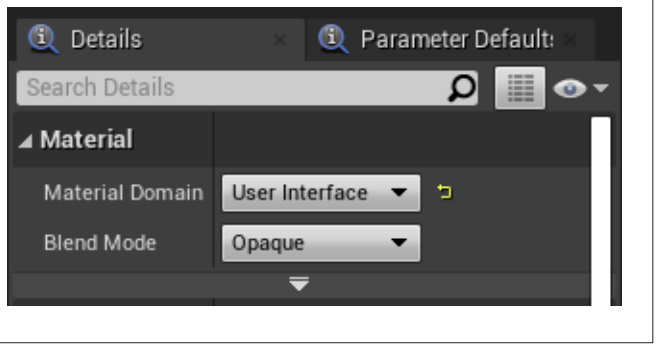
그랑프리

우선, 이전 강좌에서 배운 내용을 그대로 활용하여 게임에 메인 메뉴와 일시 정지 기능을 추가합니다. 게임의 완성도가 한층 올라갈 겁니다. 프로젝트에 레벨과 게임 모드를 선택할 수 있는 기능을 추가하려면 시간이 꽤 걸리겠지만, 그만한 가치가 있을 겁니다! 게임 모드를 추가하고 변경할 때는 주의가 필요합니다.

메뉴 시스템이 완성되면 UMG와 블루프린트로 점수, 플레이어의 생명력, 그리고 레벨의 최종 목표를 추가할 겁니다.

점수

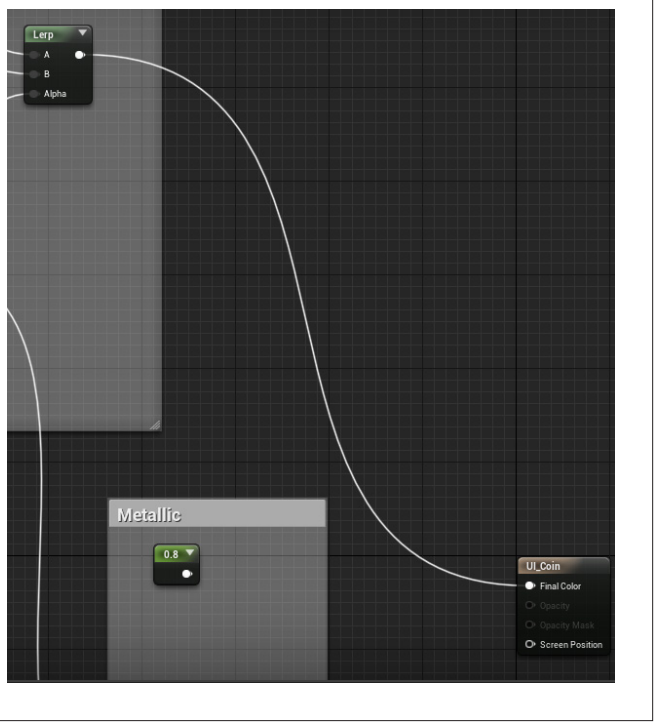
점수 시스템은 언리얼 온라인 학습 강좌에서 만든 탄약 시스템과 매우 비슷합니다. 점수는 플레이어가 획득한 코인의 수에 따라 결정됩니다. 코인을 희귀하게 해서 어려운 방법으로 레벨을 완료해야만 얻게 할 수도 있고, 레벨을 완료하기 위한 최적의 경로를 보여줄 길잡이로 쓸 수도 있습니다.



우선, 좋은 코인 이미지가 필요합니다.

1. **콘텐츠 브라우저**에서 **M_Pickup_Coin**을 검색합니다. 우클릭해서 머티리얼을 복제합니다. 새로운 머티리얼의 이름을 **M_UI_Coin**으로 설정하고 엽니다.
2. 좌측 하단 **디테일** 패널의 **Material**에 **Material Domain**을 **User Interface**로 바꿉니다. 그래프에서 **Lerp** 노드를 찾아서 출력 핀을 **M_UI_Coin** 노드의 **최종 색** 핀에 연결합니다. 머티리얼을 저장합니다.

UI 머티리얼을 생성했으니, 이제 머티리얼을 추가할 HUD를 만들어야 합니다.

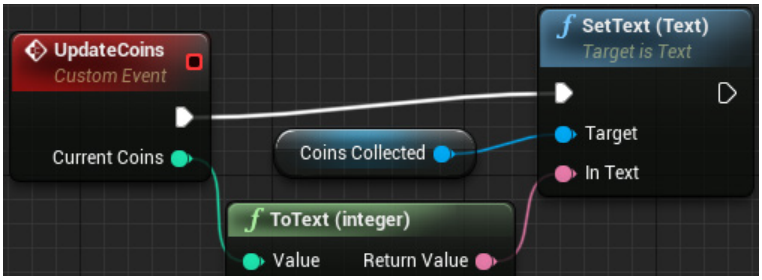


1. 다시 **콘텐츠 브라우저**로 돌아가서 강좌 내용대로 **UMG_HUD**라는 위젯 블루프린트를 만들고 엽니다.
2. Canvas Panel에 **Image**를 추가하고, **Alignment**의 **X**와 **Y**값을 **1**, **1**로 설정해서 우측 하단에 고정합니다. **Size To Content**를 체크합니다. **디테일** 패널의 **Appearance** 항목에서 **Image**를 **M_UI_Coin**로 변경하고, **Image Size**의 **X**와 **Y**값을 **160**, **160**으로 설정합니다.
3. Canvas Panel에 **Text** 위젯을 추가합니다. 이름을 **Coins Collected**로 설정하고 **변수**인지를 체크합니다. 이것도 오른쪽 하단에 고정합니다. **X 위치**를 **–65**로, **Y 위치**를 **–50**으로 설정하고 **Alignment**의 **X**와 **Y** 값을 **1**, **1**로 지정합니다. **Size To Content**를 체크합니다.
4. **Text**는 **0**으로 입력합니다. **color**는 **검은색**으로, **font size**는 **36**으로 설정합니다. **Justification**은 **중앙 정렬**로 설정합니다.

HUD의 코인 부분을 완성한 참입니다. 플레이어가 동전을 주우면 HUD가 업데이트되도록 해봅시다.

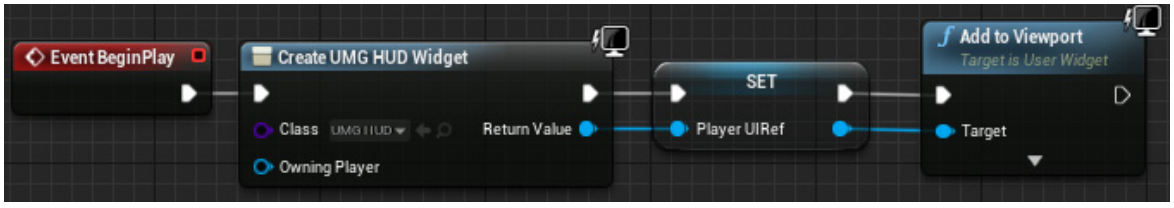
1. **UMG_HUD** 이벤트 그래프에서 **UpdateCoins**라는 이름의 커스텀 이벤트를 추가합니다.
   추가한 이벤트의 **디테일** 패널에서 **Current Coins**라는 이름의 인티저 입력을 추가합니다.
2. **Coins Collected** 텍스트 위젯의 레퍼런스를 그래프에 드래그 앤 드롭합니다.
   레퍼런스 노드에서 **Set Text** 노드를 추가합니다.
3. **UpdateCoins** 노드의 실행 핀과 **Current Coins** 핀을 **Set Text** 노드의 해당하는 핀 두 개에 각각 연결합니다.
   블루프린트를 컴파일합니다.

HUD는 장식용입니다
앞으로 알게 되시겠지만, HUD에는 많은 코딩이 필요하지 않습니다. HUD는 업데이트된 데이터를 받아서 표시만 할 뿐이고, 데이터 자체는 다른 곳에서 업데이트됩니다. HUD는 장식용이라는 원칙은 에픽의 가장 훌륭한 관행입니다. HUD는 예쁘게 보이고 플레이어에게 정보를 전달하는 것 이상은 해서는 안 된다는 뜻입니다. 데이터 저장과 연산은 게임의 다른 부분이 처리합니다.

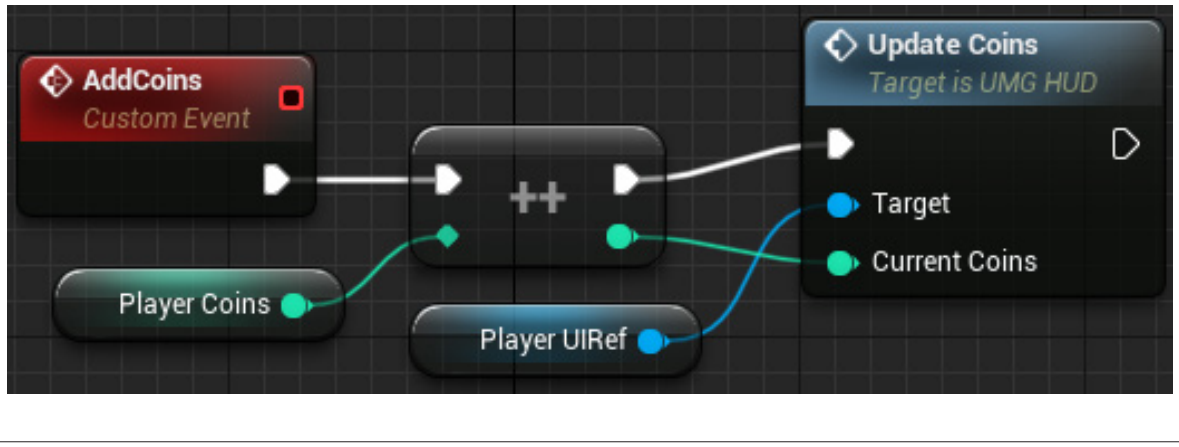


이제 **UpdateCoins** 이벤트가 발생하면 HUD가 업데이트됩니다. UpdateCoins 이벤트를 게임에 추가해 봅시다.
다음 작업은 **Game Mode** 블루프린트에서 이루어지므로, 게임 데이터와 게임 상태 정보는 계속 게임 모드에 저장하는 게 좋습니다.

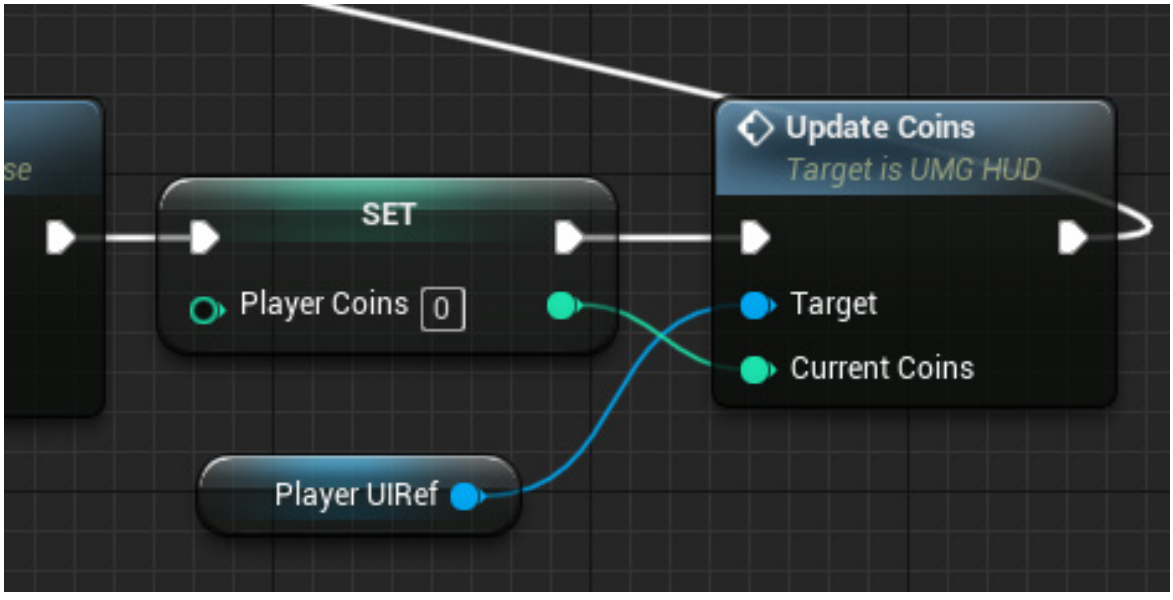
1. **이번 단계가 가장 중요합니다.** 이전 강좌와 마찬가지로 화면에 HUD부터 추가하고, 노드를 설정할 때 사용할 변수 레퍼런스를 만들 겁니다. **Game Mode** 블루프린트에서 작업하기 때문에 플레이어가 리스폰할 때 HUD를 여러 개 생성하는 문제가 일어나지 않습니다. 노드는 반드시 첫 번째 워크샵에서 만든 **Event BeginPlay** 노드와 **Bind Event to On Destroyed** 노드 사이에 추가해야 합니다. 다시 바인딩 노드에 연결하는 것도 잊지 마세요.



2. **플레이**를 눌러 뷰포트에 HUD가 제대로 표시되는지 확인합니다.
3. '**PlayerCoins**'라는 이름의 인티저 변수를 생성합니다.
4. 이벤트 그래프에 '**AddCoins**'라는 이름의 커스텀 이벤트를 생성합니다.
5. **Increment Integer** 노드를 추가합니다. '**++**'를 검색하면 쉽게 찾을 수 있습니다.
6. **PlayerCoins**의 레퍼런스를 추가하고 **Increment Integer** 노드의 초록색 입력 핀에 연결합니다.
7. **PlayerUIRef**의 레퍼런스를 그래프에 드래그 앤 드롭합니다. 레퍼런스 노드에서 **Update** 노드를 추가합니다. **Coins Update Coins** 노드의 **Current Coins** 핀을 **Increment Integer** 노드의 초록색 추가합니다. 출력 핀에 연결합니다. 그리고 실행 핀을 연결합니다.



8. 플레이어가 죽으면 **PlayerCoins** 변수가 0으로 초기화되는 기능도 있어야 합니다. 첫 워크샵에서 만든 **On Destroyed** 이벤트 끝에 **Set PlayerCoins** 변수 노드를 추가하고, **0**으로 설정합니다.
9. 그리고 앞서 했던 것처럼 **PlayerUIRef**의 레퍼런스를 그래프에 드래그 앤 드롭한 다음, **Update Coins** 노드를 추가합니다. **Current Coins** 핀을 **Set PlayerCoins** 노드의 초록색 출력 핀에 연결하고, 실행 핀을 연결합니다.



거의 다 됐습니다!

1. **콘텐츠 브라우저**에서 **BP_Pickup_Child_Coin**을 엽니다.
2. **Destroy Actor** 노드와 **Print String** 노드를 찾아서 삭제합니다.
3. **Get Game Mode** 노드를 추가합니다. 이 노드에서 **ThirdPersonGameMode**에 형변환 노드를 추가합니다. 입력 실행 핀을 **Pickup grabbed** 이벤트의 출력 실행 핀에 연결합니다.
4. 그리고 **ThirdPersonGameMode에 형변환** 노드의 **As Third Person Game Mode** 핀에서 **Add Coins** 노드를 추가합니다. 추가한 노드의 입력 실행 핀을 **ThirdPersonGameMode에 형변환** 노드의 출력 실행 핀에 연결합니다. 블루프린트를 컴파일합니다.

이제 플레이어가 획득하는 코인이 HUD 우측 하단의 총 코인 수에 추가되고, 플레이어가 죽으면 숫자가 0으로 초기화됩니다! 레벨에서 적당해 보이는 곳에 코인을 배치해 보세요!

관심사 분리
관심사 분리는 코딩의 핵심 설계 원칙입니다. 코드는 부분별로 구체적인 담당 기능이 있어야 하며, 어떤 부분이 어떤 기능을 담당하고 있는지가 분명하고 타당해야 한다는 뜻입니다.

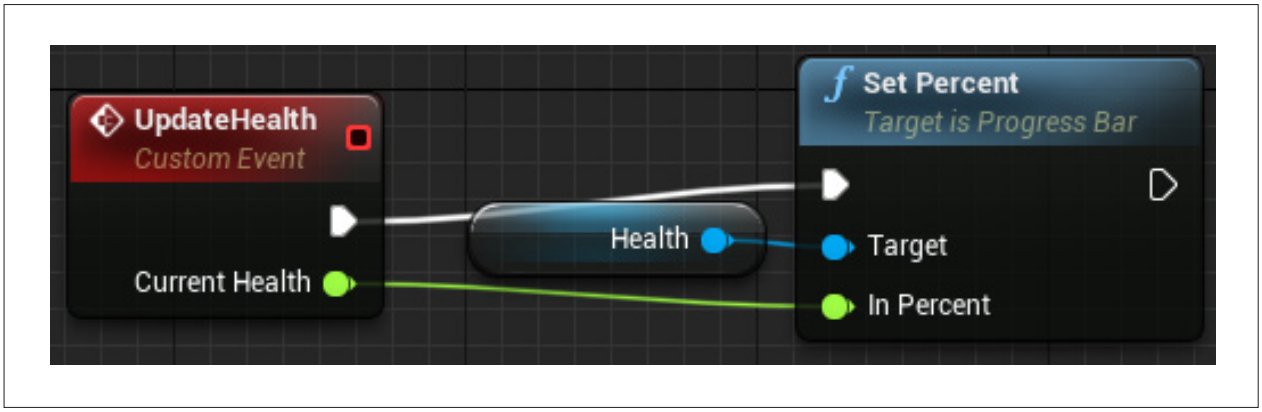
현실에서 동전을 주웠을 경우, 동전이 지금까지 동전을 몇 개 주웠는지 알려주지는 않습니다. 방금 동전을 주웠다는 사실만 알려줄 뿐입니다. 모은 동전은 지갑이 보관하고, 가지고 있는 돈의 액수는 머리가 계산합니다. 여러 기능과 관심사가 올바르게 분리된 겁니다.

관심사 분리 원칙에 맞게 게임 프로젝트를 진행하면 처음에는 작업이 더욱 복잡해 보이지만, 프로젝트 규모가 커질수록 관리가 편해집니다.

생명력

HUD에 단순한 숫자가 아닌 막대 형태로 플레이어의 생명력을 표시할 겁니다.

1. **UMG_HUD**을 엽니다.
2. Canvas Panel의 우측 하단에 고정된 **Progress Bar** 위젯을 추가하고, **디테일** 패널의 **Progress** 항목에서 **Percent**를 **100**으로 설정합니다. 위젯 이름을 **Health**로 설정하고 **변수 여부**를 체크합니다.
3. 이벤트 그래프에서 **UpdateHealth**라는 이름의 커스텀 이벤트를 추가합니다. 추가한 이벤트의 **디테일** 패널에서 **Current Health**라는 이름의 플로트 입력을 추가합니다.
4. **Health** 프로그레스 바 위젯의 레퍼런스를 그래프에 드래그 앤 드롭합니다. 레퍼런스 노드에서 **Set Percent** 노드를 추가합니다.
5. **UpdateHealth** 노드의 실행 핀과 **Current Health** 핀을 **Set Percent** 노드의 해당하는 핀 두 개에 각각 연결합니다. 블루프린트를 컴파일 합니다.

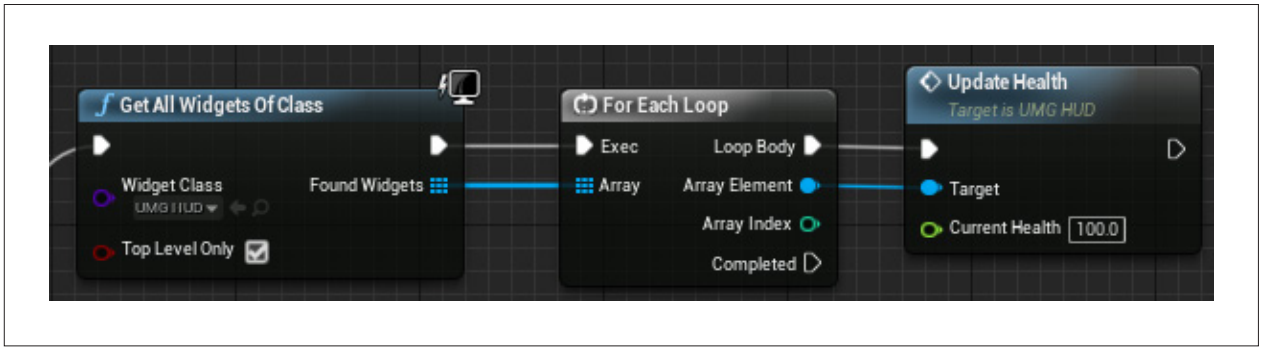


플레이어의 생명력을 UI에 업데이트하는 방법을 추가했으니, 이제 캐릭터에게 생명력을 줄 차례입니다.
다음 작업은 **ThirdPersonCharacter** 블루프린트에서 이루어집니다.

1. **PlayerHealth**라는 이름의 플로트 변수를 만들고 기본값을 **100**으로 설정합니다.
2. 그리고 **PlayerDamaged**라는 이름의 커스텀 이벤트를 생성합니다. **PlayerDamaged** 이벤트에서 **PlayerHealth**를 새로운 값으로 설정할 겁니다. 일단 **PlayerHealth**의 레퍼런스를 추가한 다음, 레퍼런스에서 **float – float** 노드를 만듭니다. 아래쪽 수(감수)를 **5**로 설정하고, **float – float** 노드의 결과를 **Set PlayerHealth** 노드에 연결합니다.
3. 이제 새로운 요령을 써볼 겁니다. **Get All Widgets Of Class** 노드를 그래프에 추가합니다. **Widget Class** 세팅을 **UMG_HUD**로 변경합니다.

Game Mode 블루프린트에서 만든 UI 레퍼런스를 대체하는 노드입니다. 특정 클래스에서 작동 중인 모든 위젯의 배열(숫자 목록)을 반환하는 편리한 기능입니다. 이 기능은 여러 UI를 한꺼번에 업데이트할 수 있게 해주며, 여러 프로페셔널 게임에서 실제로 쓰이고 있습니다.

4. **Get All Widgets Of Class** 노드의 **Found Widgets** 핀에서 와이어를 드래그하고 **For Each Loop** 노드를 추가합니다.
5. **For Each Loop** 노드의 **Array Element** 핀에서 **Update Health** 노드를 추가하고 실행 핀을 연결합니다.



A **For Each Loop** 노드는 주어진 배열에 포함된 모든 항목에 대해 일련의 노드를 작동할 것입니다. 지금은 **For Each Loop** 노드가 **UMG_HUD** 클래스의 모든 위젯을 찾습니다. 그리고 각 위젯에 대해 **UpdateHealth** 이벤트를 적용할 것입니다.
이제 제대로 된 생명력 수치를 가져오면 됩니다.

6. **UpdateHealth** 이벤트에 따라 업데이트될 **HealthProgressBar** 위젯에는 퍼센티지가 필요합니다.
   따라서 생명력 수치를 퍼센티지로 바꾸기 위해 **PlayerHealth**를 100으로 나눠야 합니다.
   Set PlayerHealth 노드의 초록색 출력 핀에서 **float ÷ float** 노드를 추가합니다. 아래쪽 수를 **100**으로 설정하고 **float ÷ float** 노드를 **Update Health** 노드의 **Current Health** 핀에 연결합니다.

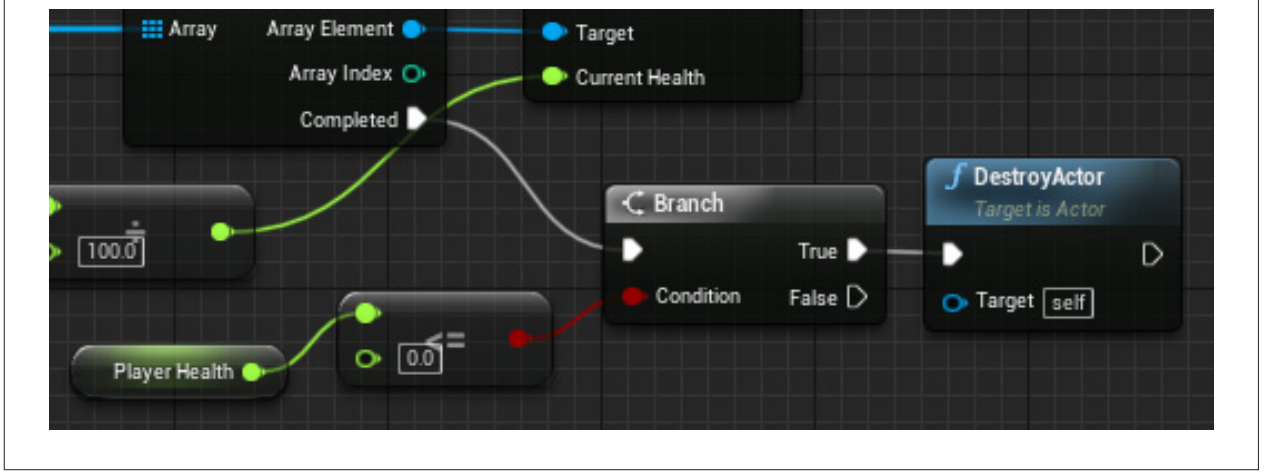
생명력이 0인데도 플레이어가 죽지 않으면 아무런 의미가 없습니다! 지금부터 추가할 기능이 그겁니다.

1. **For Each Loop** 노드의 **Completed** 핀에서 와이어를 드래그하고 **Branch** 노드를 추가합니다.

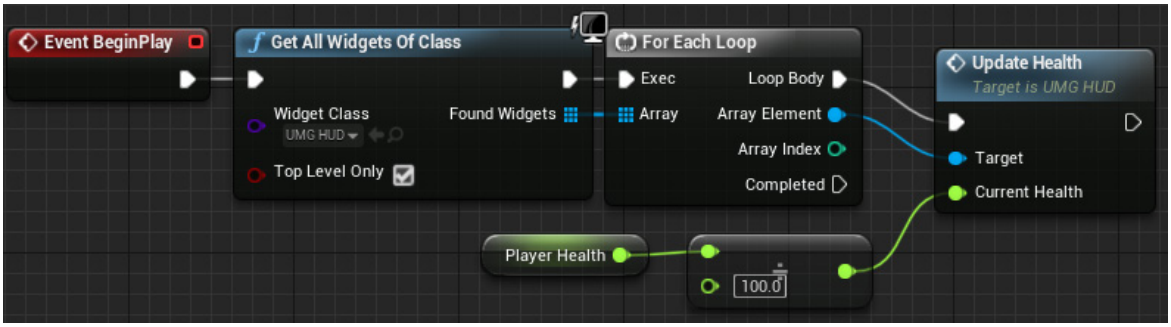
Completed 핀에 연결된 **Branch** 노드는 **For Each Loop** 노드가 배열의 모든 항목에 대해 **Loop Body** 핀에 연결된 모든 노드를 실행한 뒤 작동됩니다. 이번에는 배열의 항목이 하나뿐입니다.

2. **Branch** 노드로 **PlayerHealth**가 0 이하인지 확인해야 합니다. **Branch** 노드의 **Condition** 핀에서 **float <= float** 노드를 추가합니다. **float <= float** 노드의 상단 핀은 **Get PlayerHealth** 핀에 연결되어야 하며, 하단 핀은 **0**으로 설정되어 있어야 합니다.
3. **Branch** 노드의 **True** 핀에서 **Destroy Actor** 노드를 추가합니다. 이제 피해를 너무 많이 입으면 플레이어가 죽습니다. (그리고 리스폰합니다!)

리스폰 뒤에는 **PlayerHealth**를 **100**으로 초기화하는 것도 잊으면 안 됩니다.



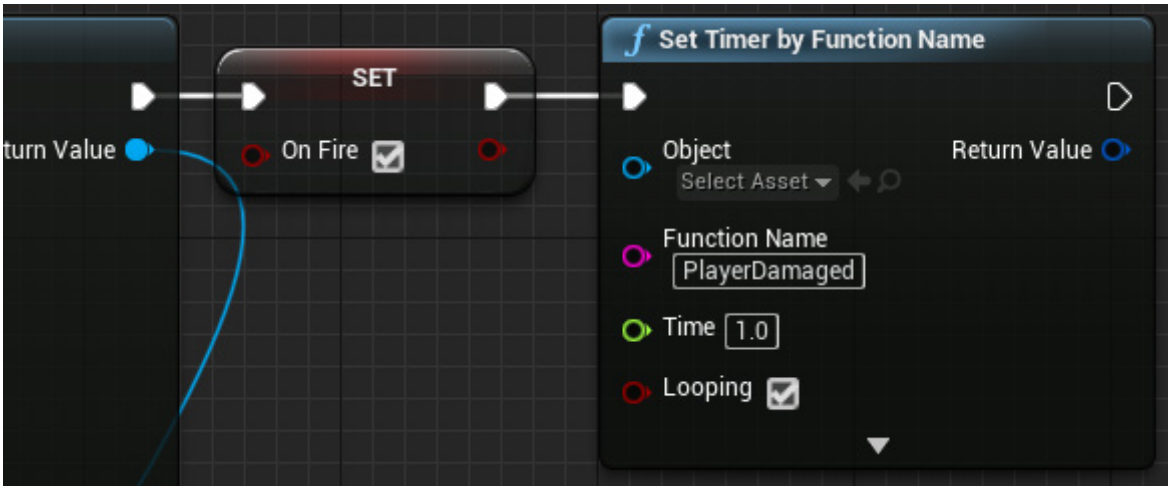
4. **Event BeginPlay** 노드를 찾습니다.
5. 위에서 했던 것처럼 **Get Widgets Of Class** 노드를 추가하고, **Widget Class**를 **UMG_HUD**로 설정한 다음 **For Each Loop** 노드를 붙입니다.
6. 이번에도 **For Each Loop** 노드의 **Array Element** 핀에서 **Update Health** 노드를 추가하고 실행 핀을 연결합니다.
7. 마지막으로 **PlayerHealth** 레퍼런스를 추가한 다음, **float ÷ float** 노드를 사용해 **PlayerHealth**를 똑같이 100으로 나누고 **float ÷ float** 노드를 **Update Health** 노드의 **Current Health** 핀에 붙입니다.



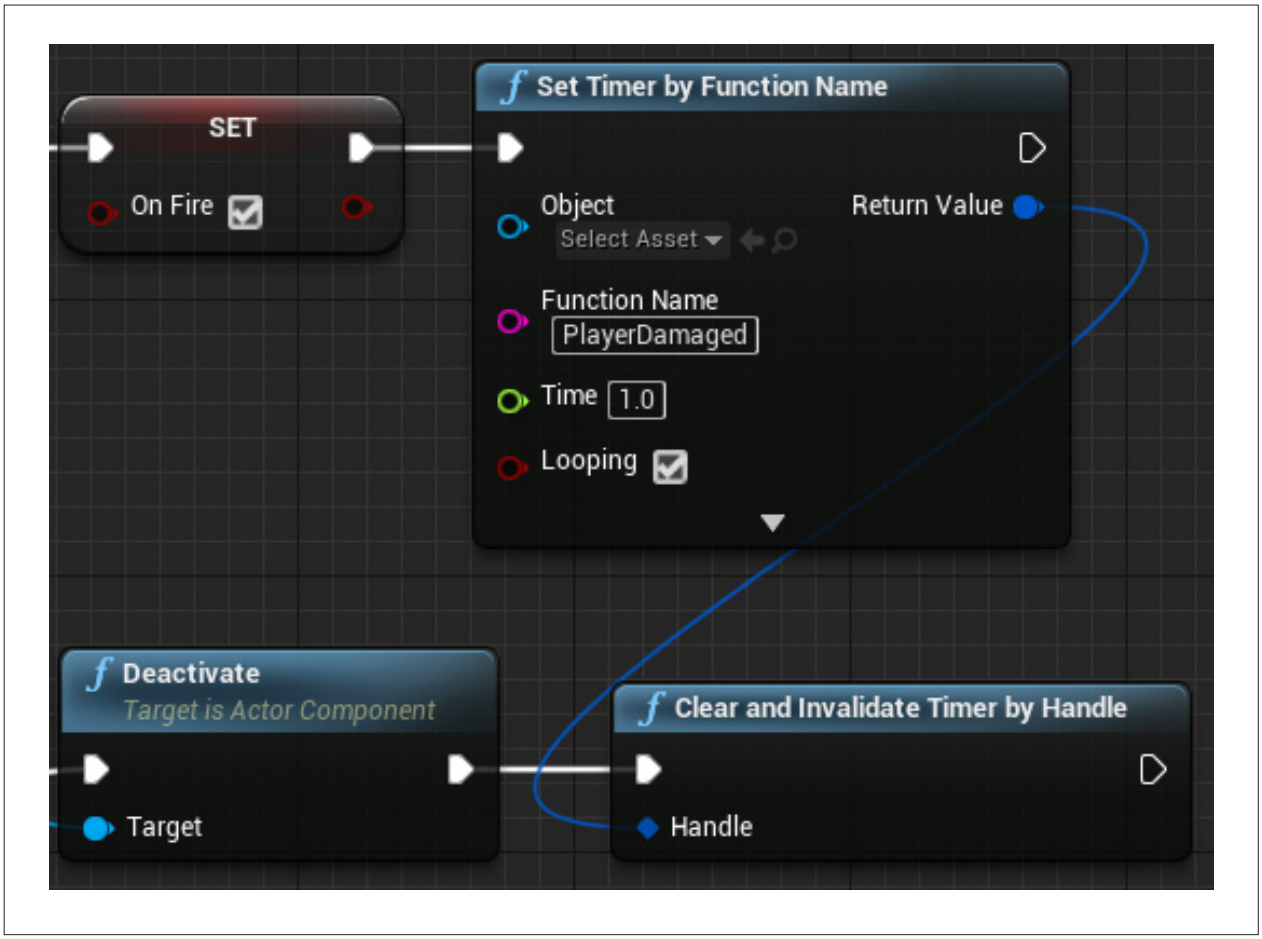
피해

플레이어가 피해를 받을 수 있게 되었으니, 이제 플레이어에게 대미지를 입힐 존재가 필요합니다! 저번 워크샵에서 사용한 불을 사용해 보겠습니다!

1. 지난 워크샵에서 플레이어에게 불을 붙일 수 있도록 만든 **On Component Begin Overlap** 이벤트를 찾습니다.
2. 체인 끝의 **Set OnFire** 노드에서 **Set Timer by Function Name** 노드를 추가합니다. **Function Name**에 **PlayerDamaged**를 입력하고 **Time**을 **1.0**으로 설정하고, **Looping**을 체크합니다.



3. **On Component End Overlap** 이벤트 끝의 **Deactivate** 노드에서 **Clear and Invalidate Timer by Handle** 노드를 추가합니다. 추가한 노드의 **Handle** 핀을 **Set Timer by Function Name** 노드의 **Return Value** 핀에 연결합니다.



이제 플레이어가 불 위에 있으면 초마다 피해를 받고, 불이 멈추면 피해도 멈춥니다.

최종 목표

최종 목표를 추가하기 위해 'UMG 입문' 프로젝트의 애셋 하나를 옮길 겁니다.

1. **Content** → **FirstHourUMG** → **Pickups**에서 **Part_LevelEnd** 캐스케이드 파티클 시스템을 찾아 우클릭합니다. **애셋 액션** → **이동**을 선택하여 파일을 그랑프리 프로젝트의 **Content** 폴더로 옮깁니다. 애셋을 옮겼으면 그랑프리 프로젝트를 엽니다.
2. 새 블루프린트 액터를 생성합니다. 이름을 **BP_LevelEnd**로 지정합니다.
3. 큐브 컴포넌트를 추가하고, **스케일** 프로퍼티를 **2.0**, **2.0**, **0.1**로 설정합니다. 소수점에 주의하세요. **Cube** 컴포넌트를 루트 컴포넌트로 만듭니다.
4. **디테일** 패널 **Materials** 항목의 드롭다운 메뉴에서 **M_Pickup_Coin**을 선택합니다.
5. 블루프린트 중앙에 **Box Collision** 컴포넌트를 추가하고, **스케일** 프로퍼티를 **1.0**, **1.0**, **20.0**으로 설정합니다.
6. 블루프린트 중앙에 **Particle System** 컴포넌트를 추가하고, **스케일** 프로퍼티를 **1.0**, **1.0**, **10.0**으로 설정합니다. **디테일** 패널의 **Particles** 아래에서 **Part_LevelEnd** 템플릿을 선택합니다. 블루프린트를 컴파일합니다.
7. **콘텐츠 브라우저**로 돌아와서 **Part_LevelEnd** 템플릿을 엽니다.
8. **이미터** 패널 왼쪽을 우클릭하고, **TypeData** → **New Mesh Data**로 이동합니다. 목록에서 Mesh Data를 선택하고, **디테일** 패널에서 메시를 **SM_Pickup_Coin**으로 바꿉니다.
9. 목록에서 **Initial Size**를 선택하고 **디테일** 열에서 **Max**와 **Min** 설정을 확인합니다. **X**, **Y**, **Z**의 **Max**를 **0.2**로, **Min**을 **0.01**로 설정합니다. **Locked Axes**는 **XYZ**를 선택합니다.
10. 템플릿을 저장합니다. 이제 블루프린트가 다양한 크기의 동전이 스폰해야 합니다.

언리얼 엔진은 거대하고 복잡한 툴입니다. 엔진 전문가도 모든 걸 완벽하게 알지는 못합니다. 언리얼 엔진으로 코딩과 개발을 할 때 중요한 건 모든 기능을 마스터하는 게 아니라 콘셉트를 구상하고, 조사와 검색을 통해 원하는 것을 구현할 수 있을 정도의 기본적 이해를 갖추는 것입니다. '정답'은 하나가 아닐 때가 많습니다.

11. **BP_LevelEnd** 이벤트 그래프에서 지금까지 배운 기능을 전부 써서 게임 승리를 위한 로직을 추가해 보세요. 플레이어, 게임 모드, UMG, 그리고 목표 사이의 상호작용을 생각해 보세요. 데이터는 어디에 저장할 건가요? 게임의 여러 요소는 어떻게 상호 작용할까요?

로직 추가 방법을 모르겠다면 팀원들과 이야기해 보고, 'UMG 입문' 프로젝트 파일을 다시 살펴보고, 다른 사람들은 어떤 방식을 사용했는지 조사해 보세요.

오프로딩 및 토론

에픽은 위젯 디자이너로 쓸 수 있는 사전 제작 위젯을 20개 이상 제공합니다. 위젯을 살펴보고 여러분의 UMG에 다양한 위젯을 추가해 보세요. 어떤 유저 인터페이스가 직관적이고 마음에 드는지 나열해 보고, UMG를 사용해 그 기능을 어떻게 다시 만들 수 있을지 생각해 보세요. 그룹과 함께 여러 게임에서 사용되는 다양한 UI에 관해 토론해 보고, 그 UI를 다시 만들 방법을 생각해 보세요. 각자 게임을 하나씩 골라서 스크린샷과 게임플레이 사례를 소개해 보세요.

마땅한 게임이 생각나지 않는다면, 기어스 오브 워의 능동적 재장전 메커니즘도 좋은 예시입니다. 에픽이 언리얼 엔진으로 개발한 기어스 오브 워는 해당 장르에 능동적 재장전 방식을 도입했습니다. 기어스 오브 워에서는 플레이어가 재장전 버튼을 누르고 진행률 표시줄이 전부 찰 때까지 기다리는 수동적 리로딩 대신에 표시줄 중간에 목표 지점을 추가했습니다. 표시줄이 목표 범위 안까지 찼을 때 플레이어가 재장전 버튼을 누르면 캐릭터가 총을 장전하는 속도가 훨씬 빨라집니다. UMG로 같은 메커니즘을 다시 만들어 볼 수 있습니다.

팀원들과 함께 좋은 예시를 살펴보면서 자신만의 UI 기능을 만들어 보고, 좋은 유저 인터페이스란 무엇인지 토론해 보세요. 팀원들이 소개한 예시들의 공통점은 무엇인가요? 플레이어의 경험에 보탬이 되는 독특한 UI 기능으로는 어떤 게 있나요?

유저 인터페이스 기능에 관해 토론하고 UMG를 사용해 일부 기능을 다시 만들거나 나만의 기능을 개발한 다음, 내가 만들 게임에 어떤 메커니즘이나 시스템을 추가하면 좋을지 아이디어가 떠오른 분도 있을 겁니다. 제작 중인 게임을 위한 소소한 아이디어를 팀원과 논의해 보세요. **규모가 아주 작은 요소를 추가해야 합니다.** 토론이 끝나면 자기가 생각해낸 메커니즘이나 시스템을 게임에 구현해 보세요. 지금까지 배운 언리얼 엔진의 다양한 시스템을 탐구해 보세요. 익숙하지 않은 부분은 조사를 통해 아이디어를 구현할 방법을 배우세요.

메커니즘이나 시스템을 추가했다면 팀원에게 보여주세요. 서로의 게임을 플레이해 보고 어떤 느낌인지 확인해 보세요! 토론할 때는 3 **D**를 중심으로 진행하세요.

- **어려움(Difficulties)**: 이번 주 워크샵에서 가장 어려웠거나 헷갈렸던 부분은 뭐였나요? 추가한 메커니즘이나 시스템의 규모가 작았나요?
- **발견(Discoveries)**: UI 요소를 만들며 어떤 걸 발견했나요? 새로 추가한 메커니즘이나 시스템을 만드는 중에 발견한 건 없었나요?
- **꿈(Dreams)**: 게임을 만드는 과정에서 떠오른 대규모 메커니즘과 시스템에 대한 아이디어는 뭐였나요? 그 아이디어를 실현하기 위해 알아야 할 건 뭔가요?

축하합니다!

패스트 트랙 워크샵3을
완료하셨습니다!

The Pathless | Giant Squid