FlipGame

Huang Bo

Exported on May 17, 2023

Table of Contents

1 游戏流程 3

1.1 游戏整体流程 3

1.2 初始化 3

1.2.1 初始化棋盘 3

1.2.2 Helper class 4

1.2.3 初始化一般放置物 5

1.2.4 初始化单位 5

1.3 AFlipPlayerState 6

1.4 棋子移动 6

1.4.1 基础移动 6

1.4.2 击退 7

1.4.3 交换两棋子 8

1.5 翻转 8

1.6 APlaceable 8

1.6.1 介绍 8

1.6.2 OnUnitOverlap 8

1.7 战斗 9

1.7.1 战斗流程 9

1.7.2 Roll点 10

1.8 道具 10

1.8.1 UItem 10

1.8.2 使用流程 11

1.8.3 道具举例 11

1.9 切换玩家 12

1.9.1 创建多玩家 12

1.9.2 输入映射 13

2 UI 14

2.1 主要按钮 14

2.2 小地图 15

2.3 玩家信息 16

2.4 UI动画 16

2.5 道具栏 17

3 Input 19

3.1 输入绑定 19

3.2 选择逻辑 19

3.3 Pick 20

3.4 具体执行 21

# 游戏流程

## 游戏整体流程

在蓝图里使用状态机来管理游戏流程



**StartGame**：初始化各种数据，包括棋盘，单位，分数，道具等。

**StartNewRound**/**StartTurn**：记录回合数，执行每轮游戏的开始动作（如：将棋子翻回其所属面）

**DecideActionPoint**：等待玩家操作（roll点），并初始化行动力

**WairForAction**：玩家的主要操作（移动/翻转)都处于这个状态

**StartBattle**/**EndBattle**：与DecideActionPoint类似，等待玩家操作（roll点），并根据结果执行获胜/失败后的操作

**StartUseItem**/**EndUseItem**：使用道具的状态，需要等待玩家选择所有target，完成后使用道具

**EndTurn**：切换到下一位玩家

Comments

Ye Shaofang

嗯另一个可以确保game state的简单机制，我想大概是把状态实际切换的函数和行为都放在GameMode的Update函数里面，并且把切换规则是否通过的判定也放在update里面。这样应该就可以避免同时触发多个gamestate了。大概类似之前ocar那里那样，实际状态切换是放在update 函数，而函数和事件只是告诉update函数有这么个请求状态切换的事情发生了。

比如，update那里有： StateSwitchable, StateRunning 两种情形。stateSwitchable的时候如果有状态切换事件就允许切换到这里；如果是StateRunning那么即使有状态切换事件也视为切换失败这样？🤔

May 15, 2023

Ye Shaofang

游戏流程其实可以考虑做一个状态锁一样的东西，这个锁或许可以通过控制玩家交互是否有效来使得游戏Game State互斥。预计我们这个版本容易出现玩家操作太快导致的bug会和玩家操作会在一个state结束之前触发另一个此时不应该触发的state。

在xx阶段里，鼠标点击视为无效

在xx阶段里，键盘操作视为无效

ue里，enhanced input action 看起来可以用来实现这种类似锁的机制。

May 15, 2023

Huang Bo

游戏的各部分逻辑（包括此状态机）目前都一齐写在GameMode里，实际上应该使用不同的manager管理不同的逻辑。

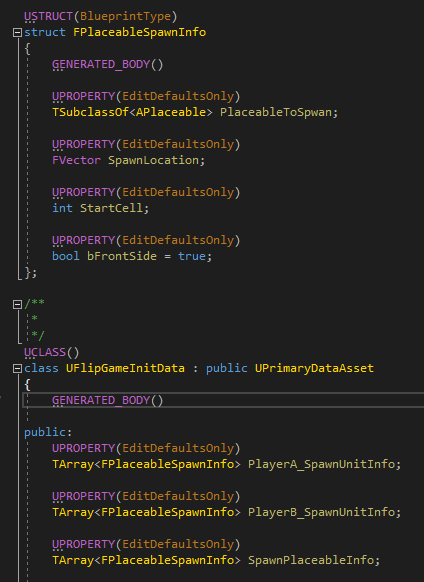
Apr 23, 2023

## 初始化

### 初始化棋盘

提供了运行时生成棋盘的功能。目前我们是固定棋盘，并在Editor里生成，无需开始游戏时生成。

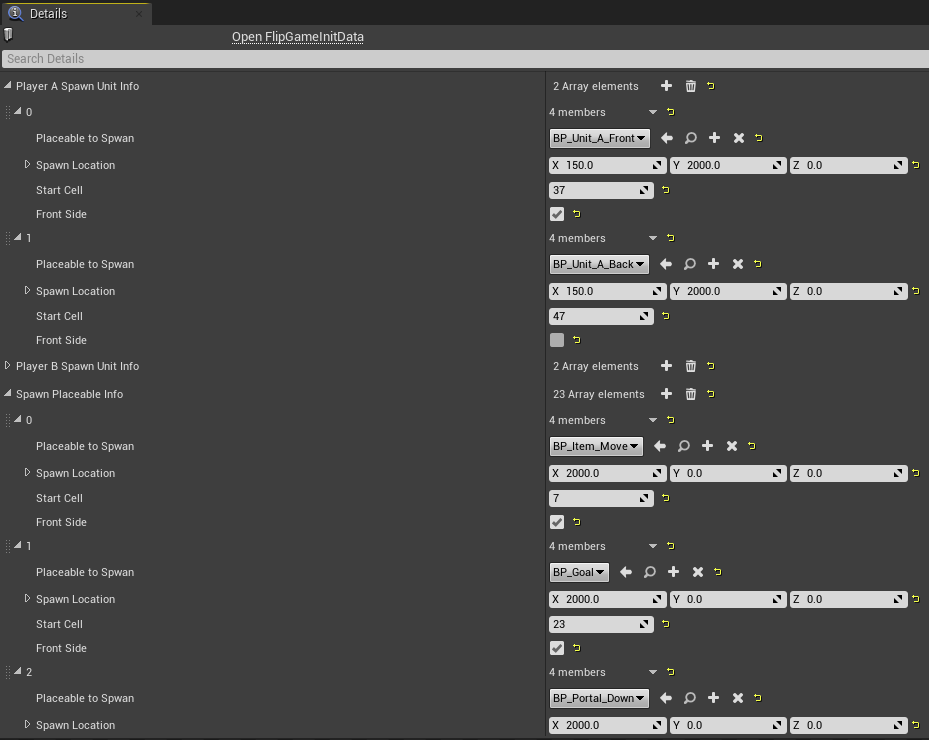
### Helper class



**FPlaceableSpawnInfo** 提供初始化一个APlaceable所属的各种信息，包括具体的类，初始化的位置，放置在棋盘上的位置和正反面信息。

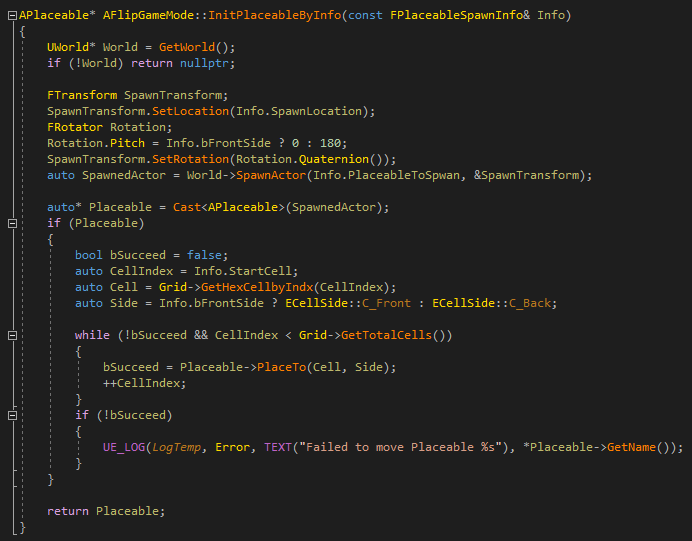
**UFlipGameInitData** 则提供三个TArray<FPlaceableSpawnInfo>， 分别是两位玩家的初始单位，和棋盘上的放置物信息。

下图是一个实际配置的例子



### 初始化一般放置物

根据从数组里拿到的初始化信息挨个生成并放置相应的类



这里会根据**bFrontSide**设置初始的Rotation，并调用**APlaceable::PlaceTo**将生成的Actor移动并Attach到相应的AHexCell上

### 初始化单位

基本与初始化一般放置物一致，区别是调用**AFlipUnit::RequestMoveUnitTo**，与APlaceable::PlaceTo直接设置位置不同，这里会有一个视觉上的移动效果，详见棋子移动部分。

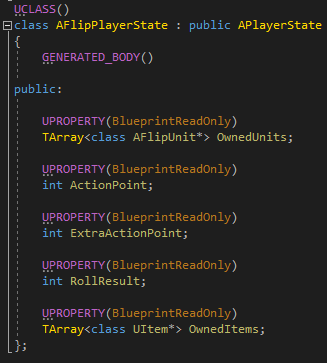
Comments

Huang Bo

1.一开始没有将地形作为初始设置，后续中翻实现了通过设置改变地形。

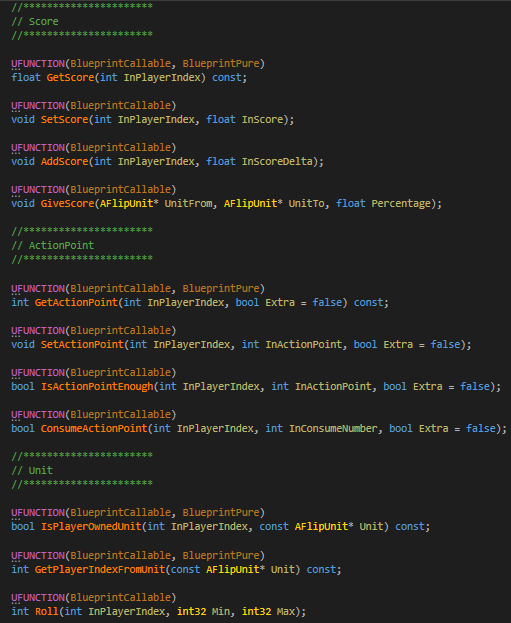
2.该设置比较抽象，无法很好的将数据和实际效果结合，后续可能需要能在editor里看到这些设置的方法。

## AFlipPlayerState



这里记录了玩家拥有的棋子，玩家的行动力，Roll点的结果，以及拥有的道具。同时使用基类里的Score来记录分数。

拥有的棋子在初始化时设置。行动力及Roll点结果通过调用AFlipGameMode里的函数来设置。



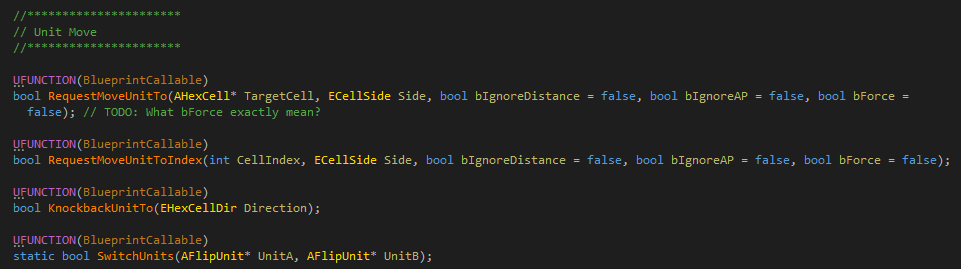
Comment

Ye Shaofang

嗯其实到现在我还在思考，棋格该不该知道每个棋子的状态呢？能不能改变每个棋子的状态呢？🤔，嗯感觉这部分其实我那边一直都没太细想，大概所以会导致后面耦合高了起来？要是有后续棋格、棋盘还有玩家操控数据那边怎么同步和交互还真特喵要想一想怎么处理

## 棋子移动

棋子提供了三种移动相关的操作



### 基础移动

**AFlipUnit::RequestMoveUnitTo**

移动前首先进行各种检查，包括地形，距离（当前只允许移动一格，之后可升级为按照AP查询），行动力

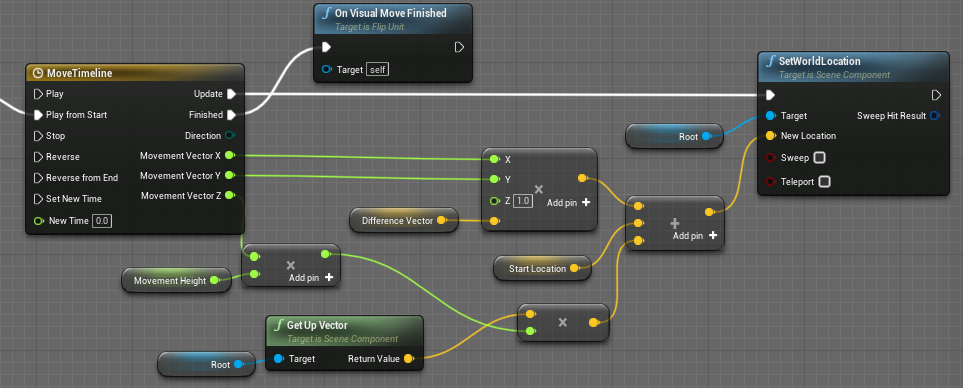
接着调用**AHexCell::EnterCell**尝试将棋子Attach到AHexCell上，成功后继续流程。

与APlaceable::PlaceTo不同，此处不会立即设置位置，而是将位置存到TQueue里，于Tick里读取内容，然后调用蓝图函数，实现视觉上的移动。这样支持了连续移动，方便后续扩展。

|  |
| --- |
| AttachToComponent(TargetComponent, FAttachmentTransformRules::KeepWorldTransform);  FVisualMoveInfo Info;  Info.TargetTransform.SetLocation(TargetComponent->GetAttachLocation());  Info.MoveType = EVisualMoveType::MOVE;  MoveInfos.Enqueue(Info); |

|  |
| --- |
| void AFlipUnit::Tick(float DeltaTime)  {  Super::Tick(DeltaTime);    if (!bIsVisualMoving && !MoveInfos.IsEmpty())  {  auto Info = \*MoveInfos.Peek();  if (Info.MoveType == EVisualMoveType::MOVE)  {  VisualMove(Info.TargetTransform);  }  else if (Info.MoveType == EVisualMoveType::SWITCH)  {  VisualSwitch(Info.TargetTransform);  }  MoveInfos.Pop();  bIsVisualMoving = true;  }  } |

蓝图里移动的视觉效果则使用Timeline来实现



### 击退

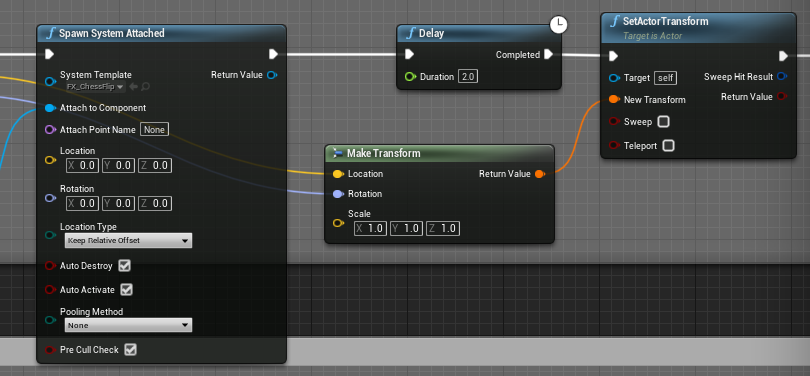
击退则是通过**AHexCell::GetNearestWalkableCellInDir**查询击退方向上合适的位置（击退方向上有障碍物则会再沿着方向再次击退直到找到合适的位置），然后调用普通的移动函数。

|  |
| --- |
| bool AFlipUnit::KnockbackUnitTo(EHexCellDir Direction)  {  if (AHexCell\* CurrentCell = Cast<AHexCell>(GetAttachParentActor()))  {  if (AHexCell\* TargetCell = CurrentCell->GetNearestWalkableCellInDir(Direction, GetSide()))  {  return RequestMoveUnitTo(TargetCell, GetSide(), true, true);  }  }  return false;  } |

### 交换两棋子

与swap函数类似，先记录下双方状态，接着调用**AHexCell::LeaveCell**和**DetachFromActor**，然后再将棋子移动到之前记录的位置。不过这里将移动类型设置为**EVisualMoveType::SWITCH**，方便蓝图与普通移动分开处理。

这里的视觉效果则是播放特效，趁特效遮挡的时候直接设置位置即可。



## 翻转

TODO

## APlaceable

### 介绍

**APlaceable**为一个基类，用于表示任何能放置在棋盘上且能动态生成/销毁/改变状态的Actor。

棋子（**AFlipUnit**），道具（**APlaceableItem**），分数（**AGoal**），传送门（**APortal**）都是继承于此。

APlaceable类提供了一些基础的查询状态的函数，销毁自己的函数，以及一个**OnUnitOverlap**，用于处理棋子移动到不同的APlaceable类上的表现。

### OnUnitOverlap

此函数用于处理一系列棋子走到不同类上的效果

**AGoal**会增加棋子和玩家的分数

|  |
| --- |
| void AGoal::OnUnitOverlap\_Implementation(class AFlipUnit\* InUnit)  {  InUnit->Score += Score;  auto GameMode = AFlipGameMode::GetFlipGameMode(this);  GameMode->AddScore(GameMode->GetPlayerIndexFromUnit(InUnit), Score);    RemoveSelf();  } |

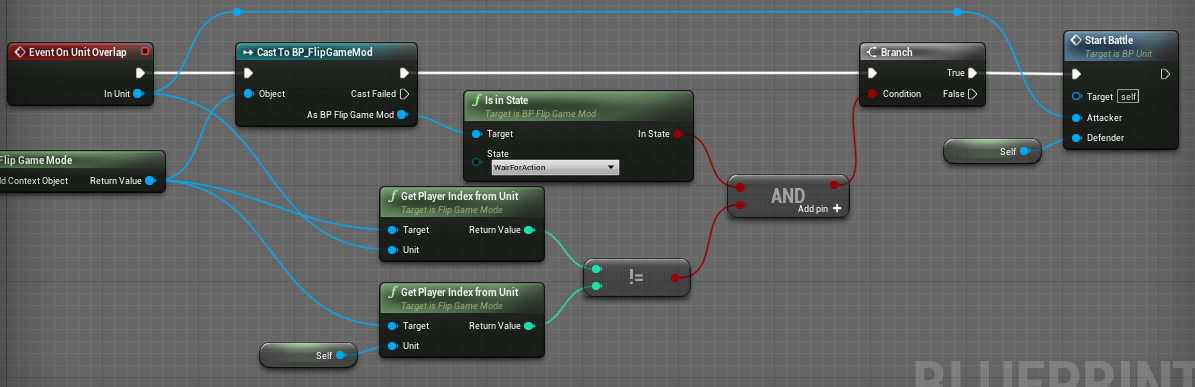
**APlaceableItem**会将道具储存起来供玩家使用

|  |
| --- |
| void APlaceableItem::OnUnitOverlap\_Implementation(class AFlipUnit\* InUnit)  {  auto GameMode = AFlipGameMode::GetFlipGameMode(this);  GameMode->StoreItem(GameMode->GetPlayerIndexFromUnit(InUnit), Item);    RemoveSelf();  } |

**APortal**则会在提供的方向上调用KnockbackUnitTo，将走上的棋子往设置的方向推动一格

|  |
| --- |
| void APortal::OnUnitOverlap\_Implementation(class AFlipUnit\* InUnit)  {  InUnit->OnFinishMove.AddDynamic(this, &APortal::UnitFinishedMove);  }    void APortal::UnitFinishedMove(AFlipUnit\* Unit)  {  Unit->OnFinishMove.RemoveDynamic(this, &APortal::UnitFinishedMove);  Unit->KnockbackUnitTo(Unit->IsInFrontFace() ? Direction : HexCellDir::GetMirroredDirection(Direction));  } |

**AFlipUnit**则在蓝图里添加Event，并进入战斗的GameFlowState。此种情况下会使移动失败，避免两棋子重叠。



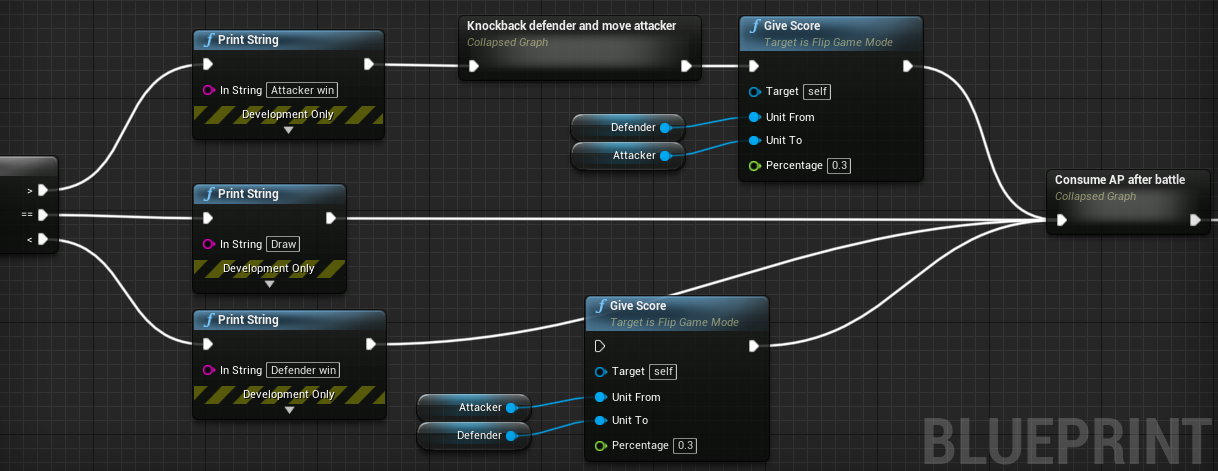
## 战斗

### 战斗流程

当棋子试图移动到对方棋子所在的格子上时，会进入战斗状态。

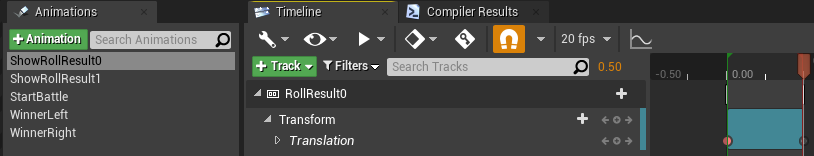
首先将State设置为StartBattle，这里会调用EventDispatcher通知UI，UI会显示双方的Roll点按钮，并隐藏其他无关按钮。

双方都完成Roll点后进入EndBattle状态，这里会检查双方Roll点的大小，根据结果执行不同的逻辑，之后回到WairForAction状态



### Roll点

玩家点击Roll点按钮时，会调用**AFlipGameMode::Roll**更改PlayerState数据，然后将返回值设置到UI显示。并且会播放简单的UI动画，让结果从屏幕外移动到屏幕内。



根据当前的GameFlowState，Roll点的结果用做不同的情况。DecideActionPoint则会调用**AFlipGameMode::SetActionPoint**设置行动力。

若在战斗状态，则会在检测到双方玩家均Roll完点后，进入EndBattle状态

## 道具

### UItem

UItem类使用TArray<UObject\*>来存储需要的信息，称为**Targets**。并提供函数FillTarget来添加Target，同时提供IsValidTarget来筛选。

当Targets数量达到设定的TargetNum后，则会触发Delegate通知。AFlipGameMod里收到通知则会调用Use来使用道具。

道具是储存在APlaceableItem里，蓝图会继承APlaceableItem设置不同的道具。因此，我们需要在UPROPERTY里加上**Instanced**来设置，并且在类的UCLASS里加上**EditInlineNew**来编辑。

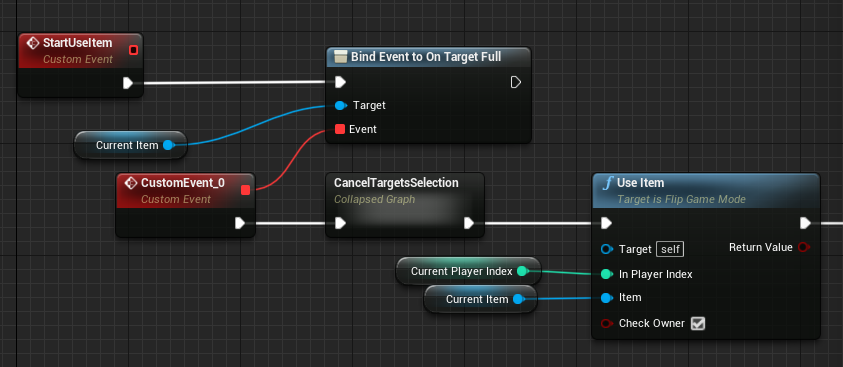
|  |
| --- |
| UCLASS()  class APlaceableItem : public APlaceable  {  GENERATED\_BODY()    public:  virtual void OnUnitOverlap\_Implementation(class AFlipUnit\* InUnit) override;    UPROPERTY(EditAnywhere, BlueprintReadWrite, Instanced)  class UItem\* Item;  }; |

|  |
| --- |
| UCLASS(Blueprintable, BlueprintType, EditInlineNew)  class UItem : public UObject  {  //...  }; |

### 使用流程

当玩家进入APlaceableItem所在的格子时，会调用**AFlipGameMode::StoreItem**将道具存储在**AFlipPlayerState::OwnedItems**里，同时会创建相应的UI按钮显示在屏幕上。

当玩家点击相应的UI按钮后，会进入StartUseItem状态，此时会绑定OnTargetFull Delegate。在该状态下，玩家的所有点击操作都会拿到相应的Actor或者Component来调用FillTarget，在该函数内会调用IsValidTarget来筛选合法的Target。



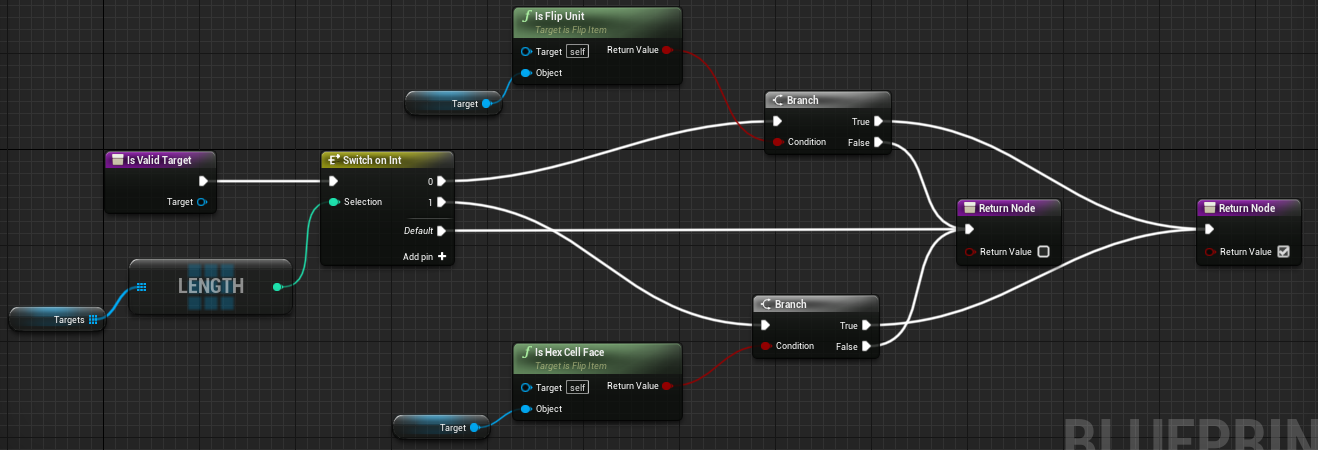
当Targets达到设定数量后，收到OnTargetFull的通知，则会使用道具，并进入EndUseItem状态。若是玩家取消使用，则会直接进入EndUseItem。EndUseItem会清除所有Target，然后进入WairForAction状态。（这里应该解绑Delegate）

### 道具举例

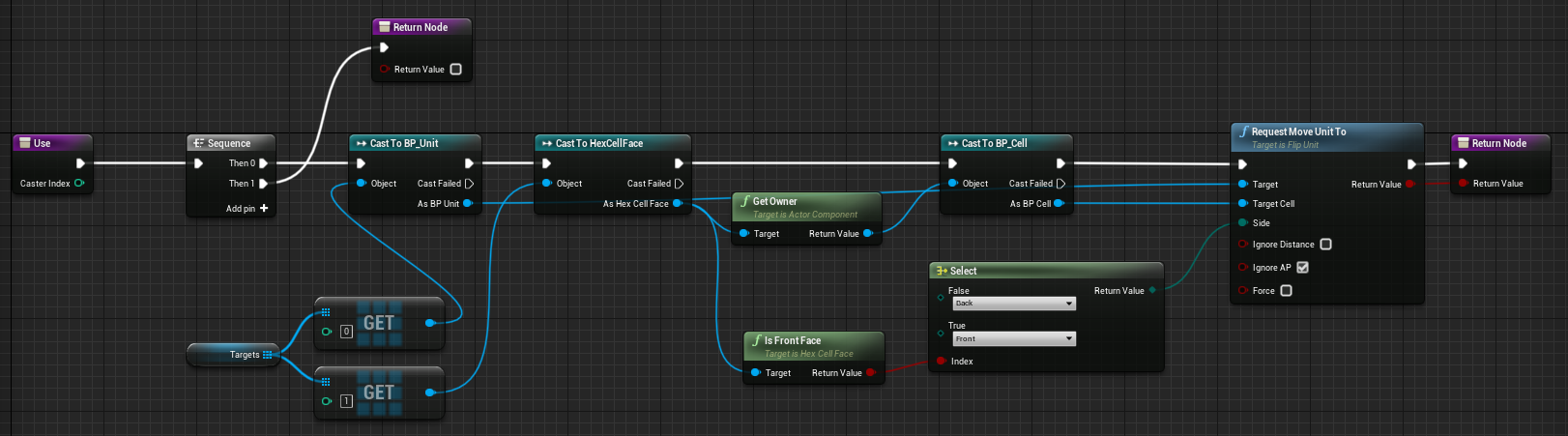
游戏共有三种道具

**1.移动棋子**

从道具需要玩家选择一个棋子，和一格内的格子。在IsValidTarget里检查是否是需要的类型



在使用时则调用AFlipUnit::RequestMoveUnitTo函数来移动棋子



**2.交换棋子**

与移动类似，只是Target为两个不同的棋子，并且调用AFlipUnit::SwitchUnits来交换

**3.复制地形**

Targets为两个不同的地块（UHexCellFace）。

设计时要求能够复制转送们，故使用时会先检查是否为传送门，若是，则在目标地块上生成传送门。若否，则调用AHexCell::ChangeTerrainType来改变地形。

## 切换玩家

回合结束时，会根据当前玩家的ID，将状态设置为StartNewRound/StartTurn。除此之外，还要改变当前玩家。

为了避免在游戏中多次记录并检查当前玩家 ，我采用了Local Multiplayer的做法，生成两个玩家，并关掉分屏。这样也便于将游戏做成联机游玩。

参考：

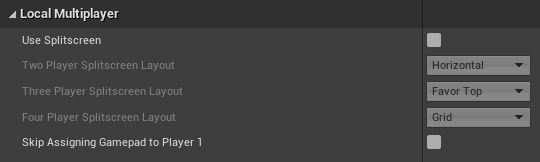
<https://unrealcommunity.wiki/local-multiplayer-tips-993f4t24>

### 创建多玩家

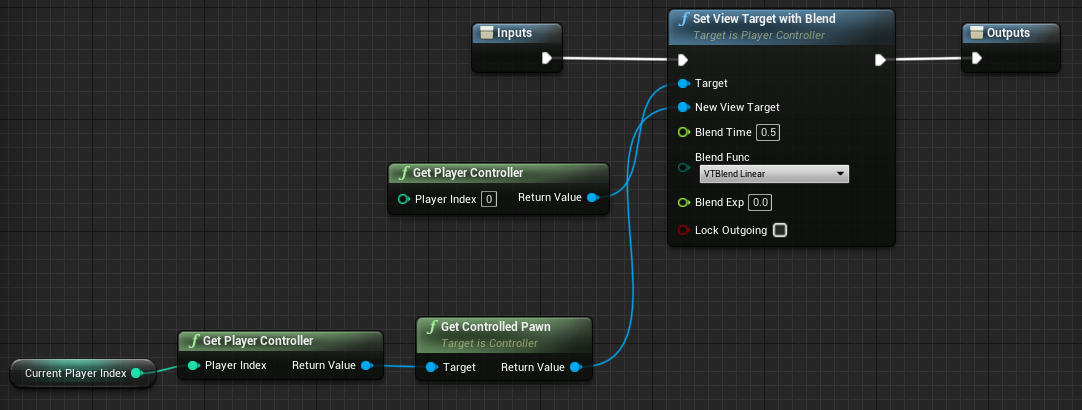
在**AFlipGameMode::BeginPlay**里根据设置的玩家数量（目前为2）生成创建新玩家

|  |
| --- |
| void AFlipGameMode::BeginPlay()  {  Super::BeginPlay();    for (int i = 1; i < TotalPlayerNumber; ++i)  {  UGameplayStatics::CreatePlayer(this);  }  } |

我们只需要一个视角，故在设置里关掉分屏



在游戏中也能切换各玩家自己的摄像机



### 输入映射

引擎默认的本地多人操作为键鼠对应玩家0，多个手柄对应接下来的玩家。

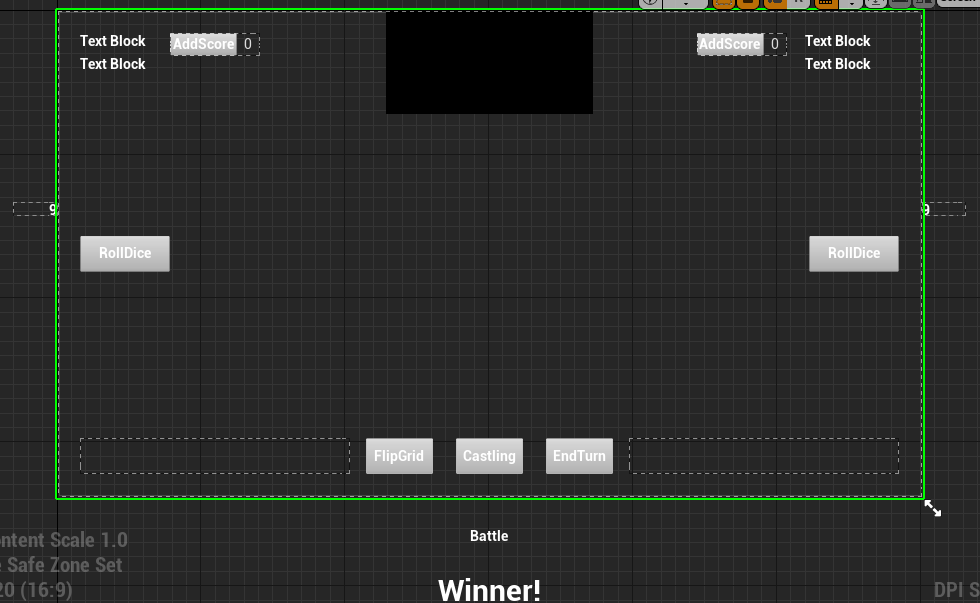
我们游戏仅用鼠标操作，故需要在切换到第二个玩家时，将输入也映射到第二个玩家。

为此，我们创建自己的UFlipViewportClient类，继承自UGameViewportClient，并且override InputKey和InputAxis函数。

在我们的函数里，从AFlipGameMode拿到当前玩家，并将参数ControllerId相应的增加，以此来让输入偏移到相应的玩家上。

|  |
| --- |
| bool UFlipViewportClient::InputKey(const FInputKeyEventArgs& EventArgs)  {  FInputKeyEventArgs NewEventArgs = EventArgs;  if (World && !EventArgs.IsGamepad() && EventArgs.ControllerId == 0)  {  if (auto GameMod = World->GetAuthGameMode<AFlipGameMode>())  {  NewEventArgs.ControllerId+= GameMod->GetCurrentPlayerIndex();  }  }    return Super::InputKey(NewEventArgs);  }    bool UFlipViewportClient::InputAxis(FViewport\* InViewport, int32 ControllerId, FKey Key, float Delta, float DeltaTime, int32 NumSamples /\*= 1\*/, bool bGamepad /\*= false\*/)  {  if (World && !bGamepad && ControllerId == 0)  {  if (auto GameMod = World->GetAuthGameMode<AFlipGameMode>())  {  ControllerId += GameMod->GetCurrentPlayerIndex();  }  }    return Super::InputAxis(InViewport, ControllerId, Key, Delta, DeltaTime, NumSamples, bGamepad);  } |

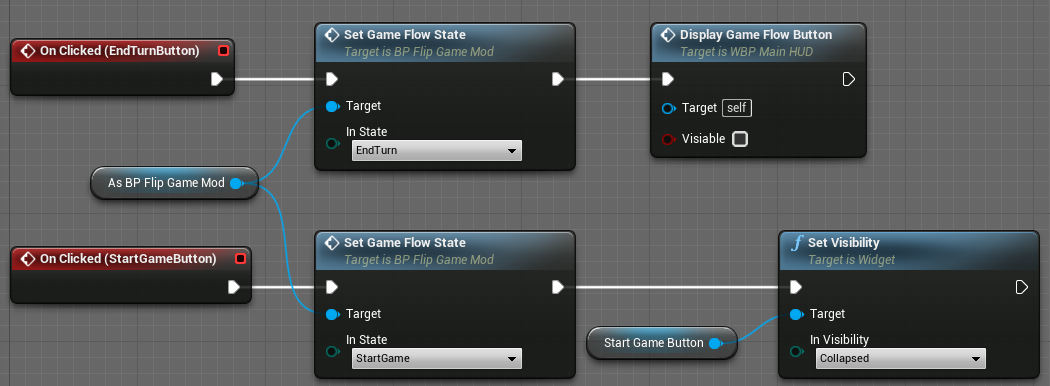
# UI



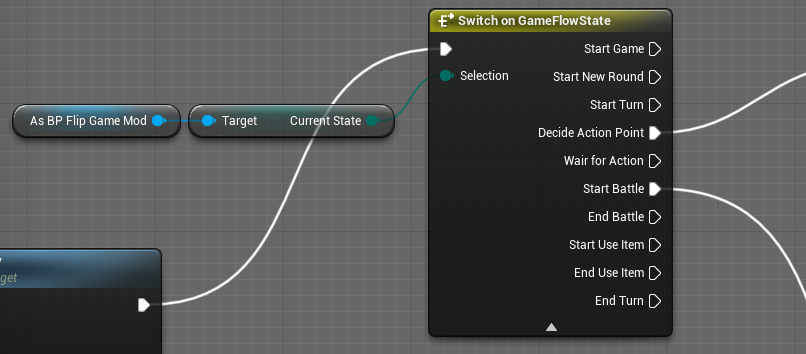
## 主要按钮

提供了几个按钮来对应游戏功能，包括结束回合，翻转整个棋盘，棋子互换和roll点。

该UI在初始化时会绑定BP\_FlipGameMod里的Event，当状态变化时，需要改变各按钮的显示/隐藏。如：Roll点时需要隐藏结束回合/互换的按钮，防止如跳过State流程等错误。



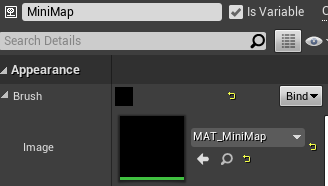
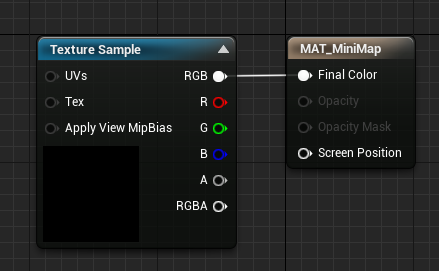
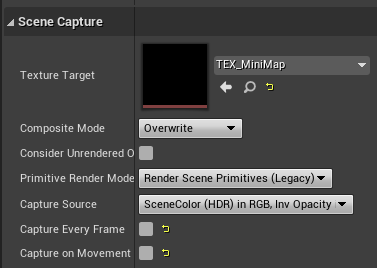
除Roll点外其他按钮均只对应一个功能，比较简单。Roll点目前有两种情况，一是在DecideActionPoint，一种是在StartBattle，需要分开处理。



## 小地图

正上方的image用来展示棋盘背面，方便玩家观察

使用**USceneCaptureComponent2D**来渲染场景到RenderTarget贴图上，然后设置Material Domain为UI的简单材质，赋予image即可。



注意USceneCaptureComponent2D的bCaptureEveryFrame默认为True，如场景里有多个，需要设为false然后手动管理。

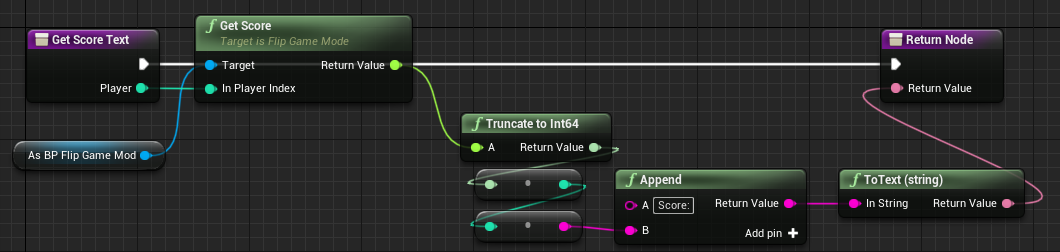
可参考以下链接设置

<https://nerivec.github.io/old-ue4-wiki/pages/umg-mini-map.html>

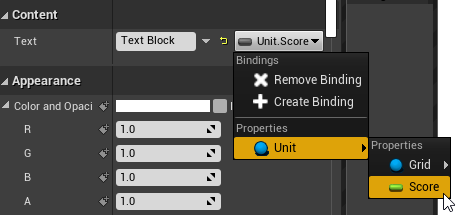
## 玩家信息

左（右）上角会显示玩家的分数以及行动力

使用function binding来实时更新分数



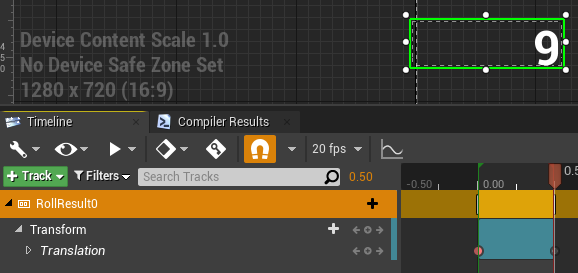
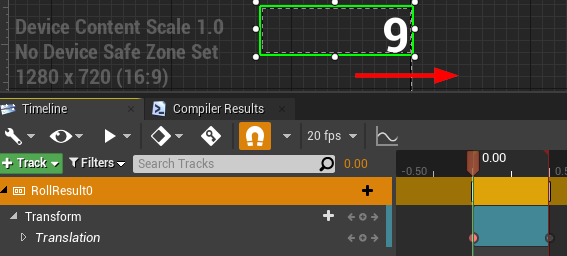
棋子的分数则使用**UWidgetComponent**，其不需要额外文字，直接绑定相关的数据



## UI动画

当玩家roll点/战斗/最终胜利时，需要有文字出现提示玩家，于是使用了简单的UI动画

动画仅改变Text的位置，通过PlayAnimationForward和PlayAnimationReverse来让文字进入/离开屏幕

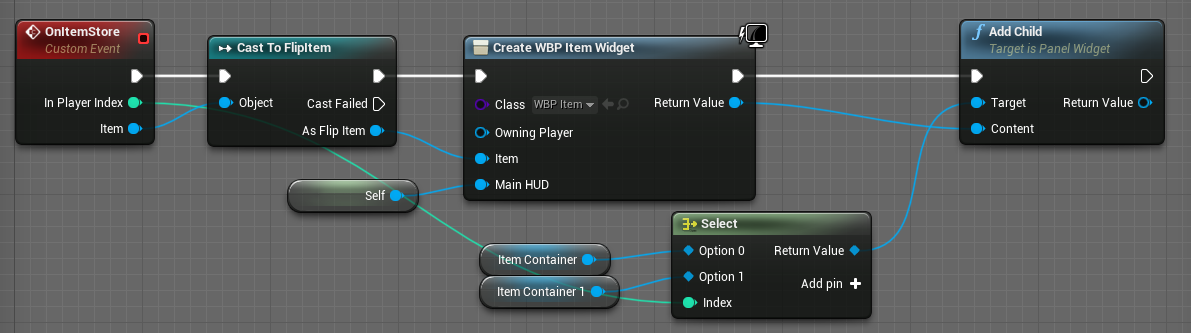


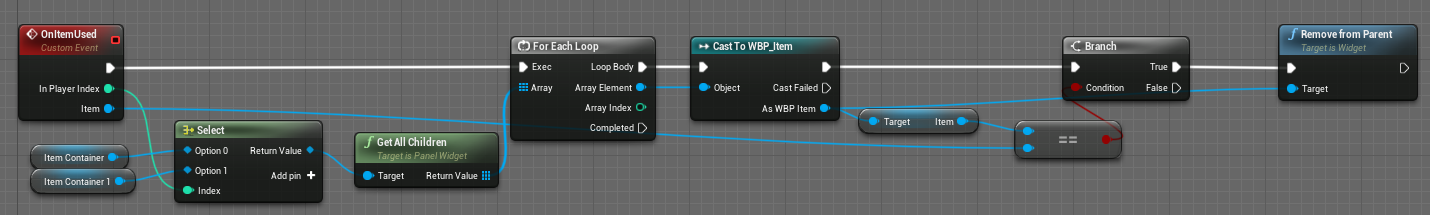
## 道具栏

使用单独的widget来表示道具，其仅为一个按钮



在主UI里，使用UHorizontalBox作为container，动态添加/删除道具。之后可以考虑重复利用而不是直接删除。





Comment

Huang Bo

应该将几处元素结合成小的widget（例如玩家的信息，道具栏等），然后双方玩家就可以复用。但是需要左右排列顺序互换，这个方法还需要研究。

# Input

AFlipPlayerController

## 输入绑定

目前操作仅支持鼠标，故只绑定4个Event

|  |
| --- |
| InputComponent->BindAction("Confirm", IE\_Pressed, this, &AFlipPlayerController::OnConfirmPressed);      InputComponent->BindAction("Confirm", IE\_Released, this, &AFlipPlayerController::OnConfirmReleased);      InputComponent->BindAction("Cancel", IE\_Pressed, this, &AFlipPlayerController::OnCancelPressed);      InputComponent->BindAction("Cancel", IE\_Released, this, &AFlipPlayerController::OnCancelReleased); |

在Tick里记录鼠标按下的时长，然后Release的时候进行检测，以识别是点击还是长按

|  |
| --- |
| void AFlipPlayerController::OnConfirmPressed(FKey key)  {      bConfirmPressed = true;  }    void AFlipPlayerController::OnConfirmReleased(FKey key)  {      bConfirmPressed = false;      if (IsClick(ConfirmPressedTime))      {          //...      }      ConfirmPressedTime = 0.f;  }    bool AFlipPlayerController::IsClick(float Time)  {      return Time < ClickTimeThreshold;  }    void AFlipPlayerController::PlayerTick(float DeltaTime)  {      Super::PlayerTick(DeltaTime);        if (bConfirmPressed) ConfirmPressedTime += DeltaTime;      if (bCancelPressed) CancelPressedTime += DeltaTime;  } |

## 选择逻辑

这里假设了三种选择状态

|  |
| --- |
| UENUM()  enum class ESelectionState : uint8  {      NONE,      UNIT,      CELL,  }; |

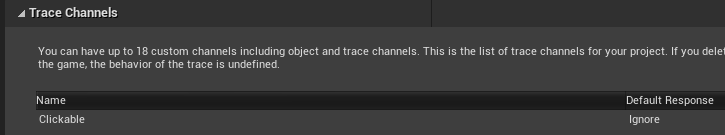
根据不同的状态，在单击左键时选择调用不同的逻辑

|  |
| --- |
| if (CurrentSelectionState == ESelectionState::NONE)          {              SelectUnitOrCell();          }          else if (CurrentSelectionState == ESelectionState::UNIT)          {              MoveUnit();          }          else if (CurrentSelectionState == ESelectionState::CELL)          {              FlipOrSelect();          } |

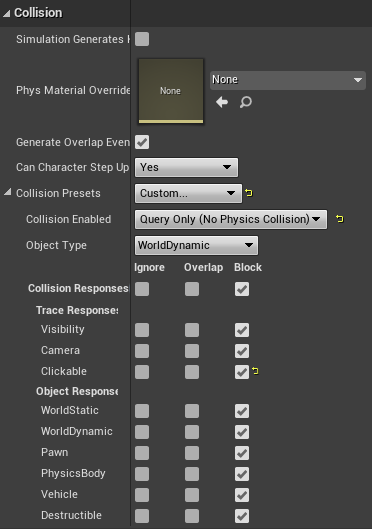
单击右键则为取消所有选择

## Pick

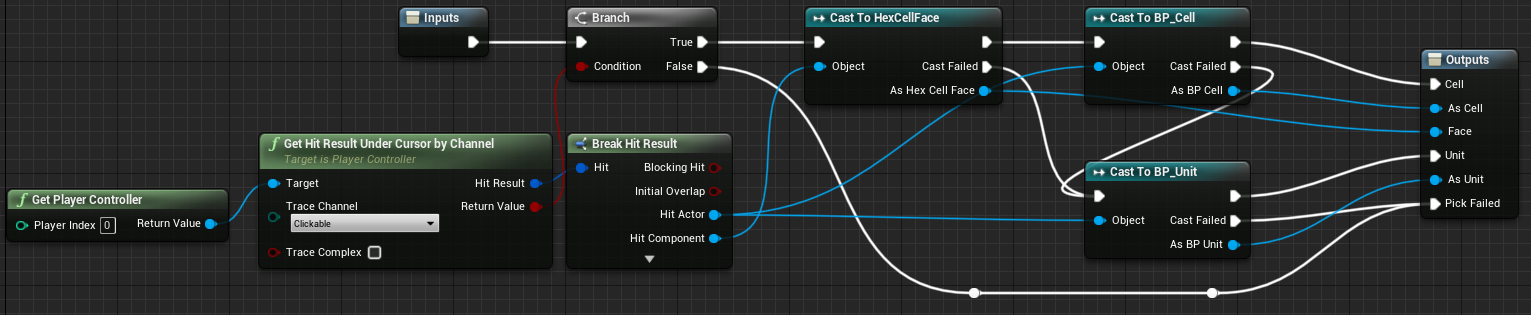
我们只允许选中棋子和棋盘格，故在**Project Settings**的Collison里添加了额外的**Trace Channel**



在棋子和棋盘格的mesh里，设置**CollisionResponses**对Clickable为Block。场景里所有其他mesh保持默认为Ignore，则玩家就只会选择棋子和棋盘格。



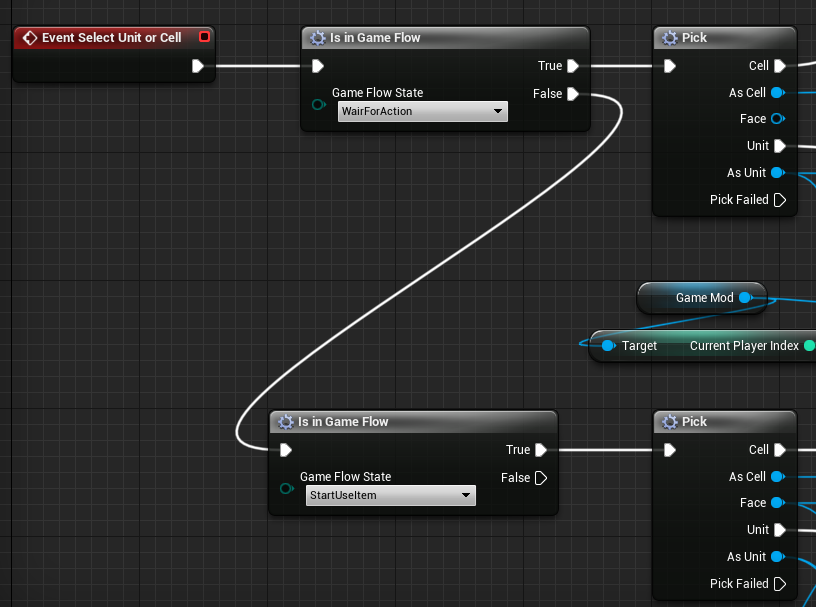
因为游戏目前不需要物理碰撞，故将**CollisionEnabled**设为QueryOnly  
这样设置，玩家点击鼠标后，便可以使用**GetHitResultUnderCursorByChannel**来检测是否点到相应的物体上。后续则根据选的物体不同执行不同的逻辑。



注意这里只能使用 PlayerController 0 来做检测

## 具体执行

**SelectUnitOrCell**  
根据当前状态执行不同的逻辑



若是WairForAction则直接选中。将选中的单位记录下来，并通知选中，执行高亮等逻辑。  
若是StartUseItem，则调用相应的FillTarget函数。  
**MoveUnit**  
若选中的是棋格则可以直接移动。如果是棋子，则需要先获取棋子所在的格子，然后调用移动（视情况进行战斗或者移动失败）。  
**FlipOrSelect**  
若玩家选中某格两次，则执行翻转。若玩家选择的是其他格子/单位，则重新调用SelectUnitOrCell。

Comment

Huang Bo

由于一开始只考虑了几种情况，也缺乏对于成熟Input设计的了解，导致后续逻辑加的非常混乱，需要重新思考如何实现。