Experience System

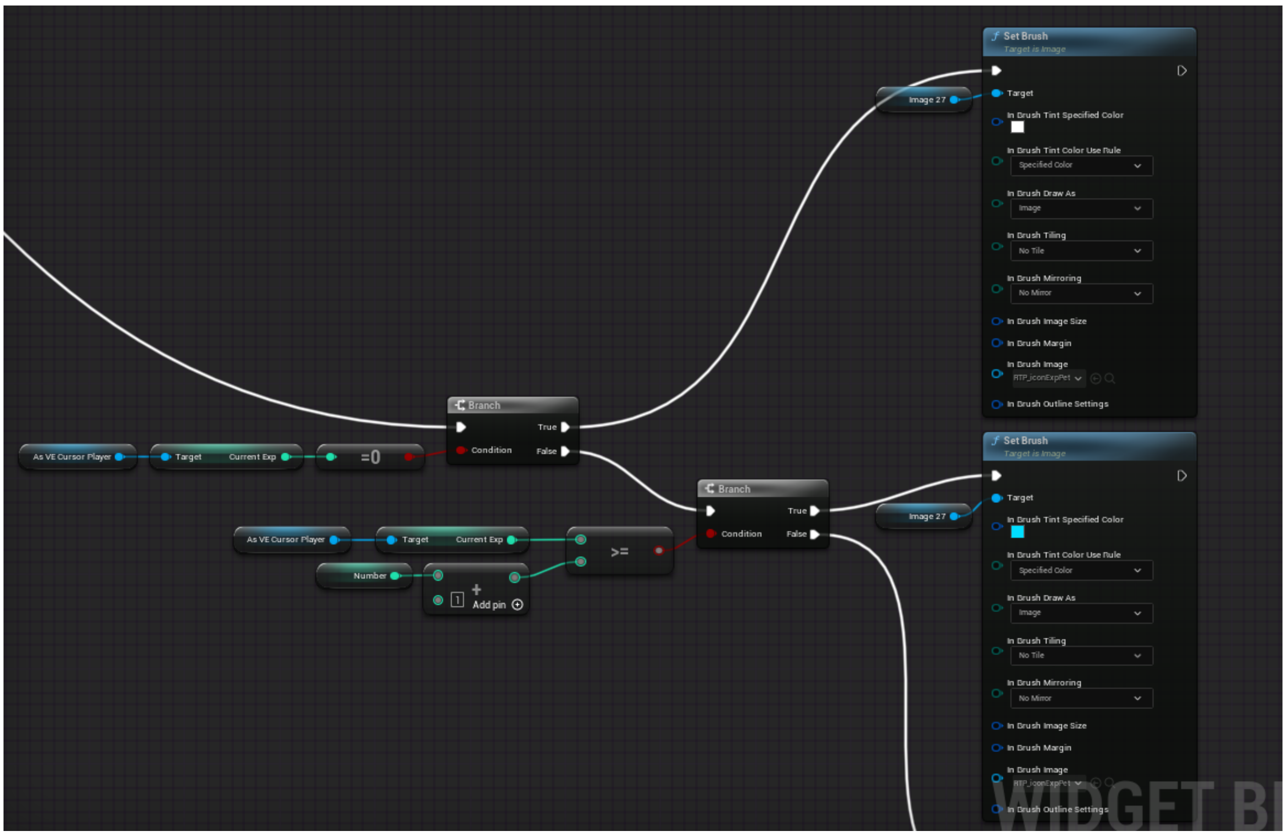
Information on how the level up and experience for the player is implemented into the project.

UI

How the UI gets and displays Levels/Experience

UI

Clock Experience

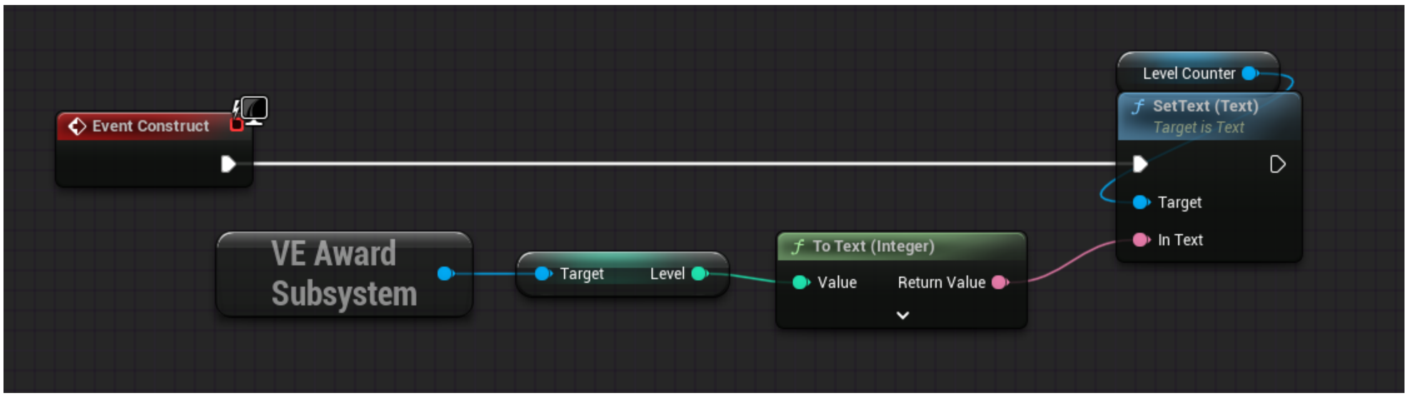


The clock experience display is fairly simple, it gets updated with a event dispatcher on the Award Subsystem. The function above can be found in RTP_ClockPettel. (This has slightly changed, its the same but it plays animations)

UI

Level Display Widget

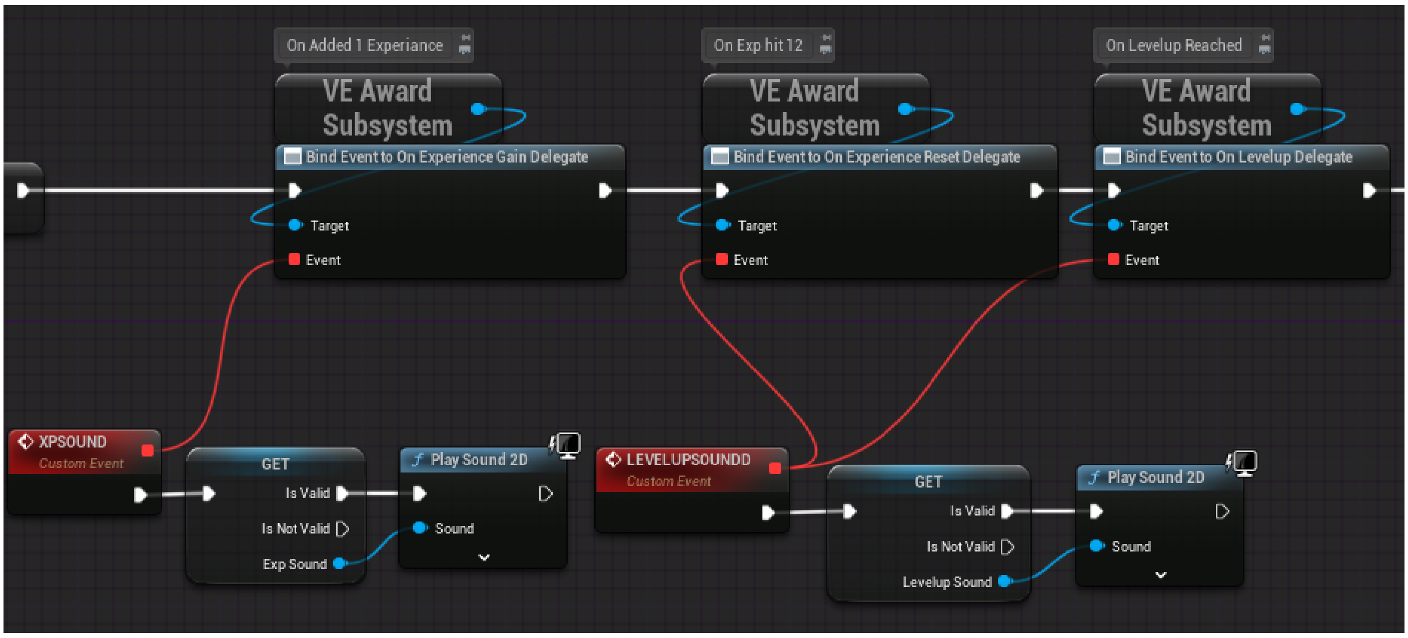
VE_LevelDisplay



Cursor Player

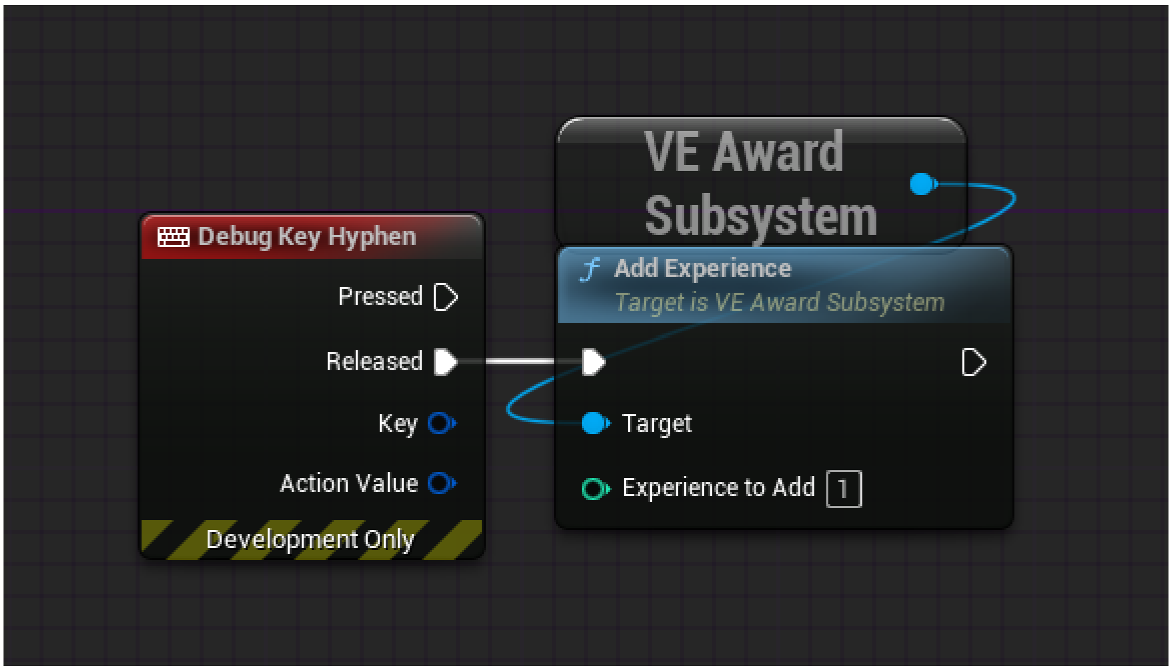
How the player works with and stores the Levels/Experience

Playing Exp and Levelup sounds



On begin play the player binds to the events on the Award Subsystem and plays the sounds when called

Add Experience Debug Key

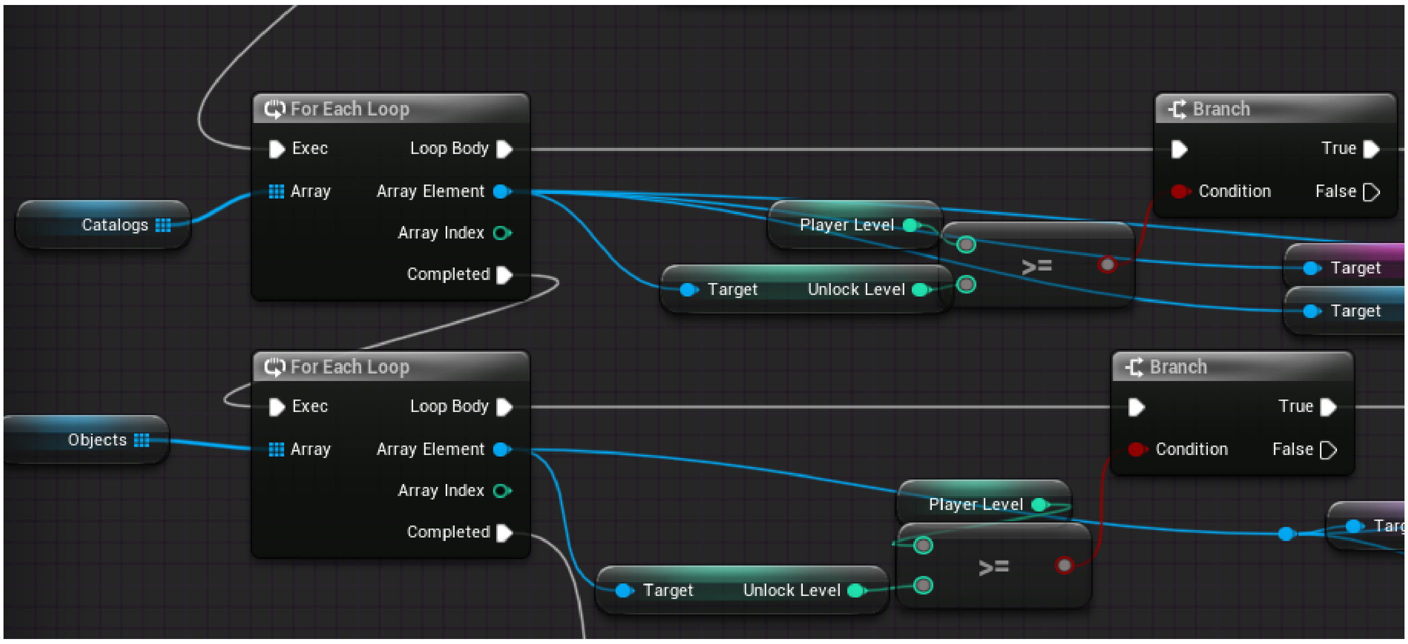


The player has a debug key for giving yourself a xp point when you press "-"

Shop Items and Categorys

How shop wheel knows to display items based on Level

Simple Level check while adding entry's to the shop

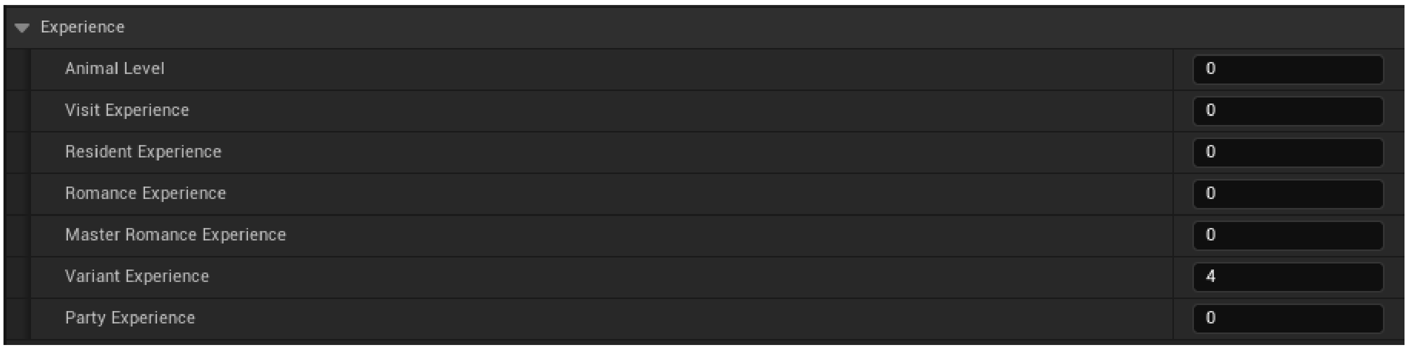


This is located in the function "M_LoadCatalog" inside of "ShopWheelController" (Rather then getting the level from the player it now gets it from the Award Subsystem)

DataAssets

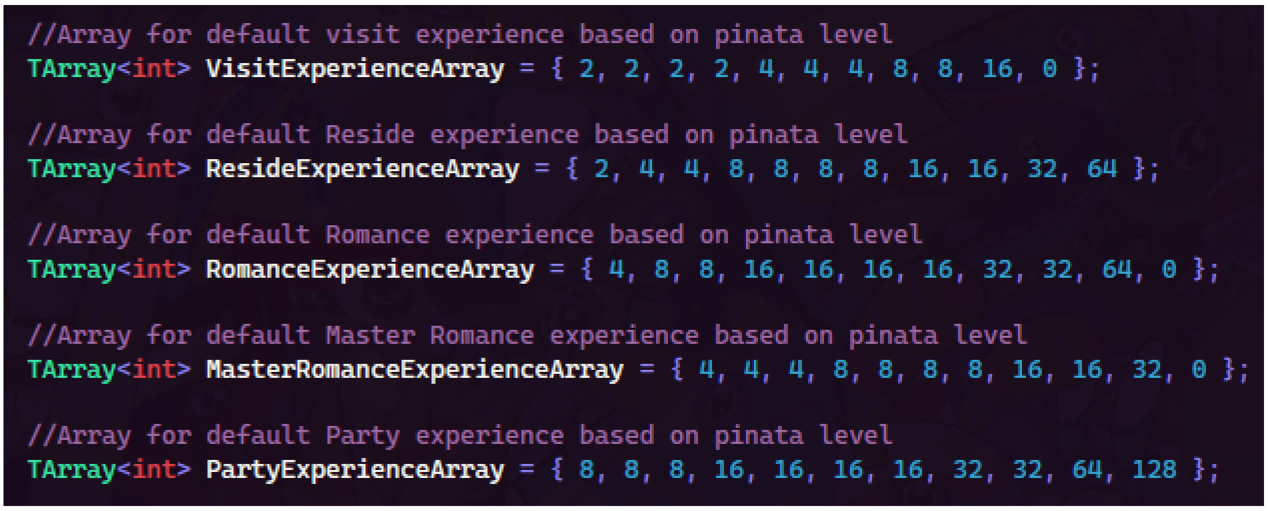
Where are the Experience parameters stored?

Animal Definition Data



Experience	
Animal Level	0
Visit Experience	0
Resident Experience	0
Romance Experience	0
Master Romance Experience	0
Variant Experience	4
Party Experience	0

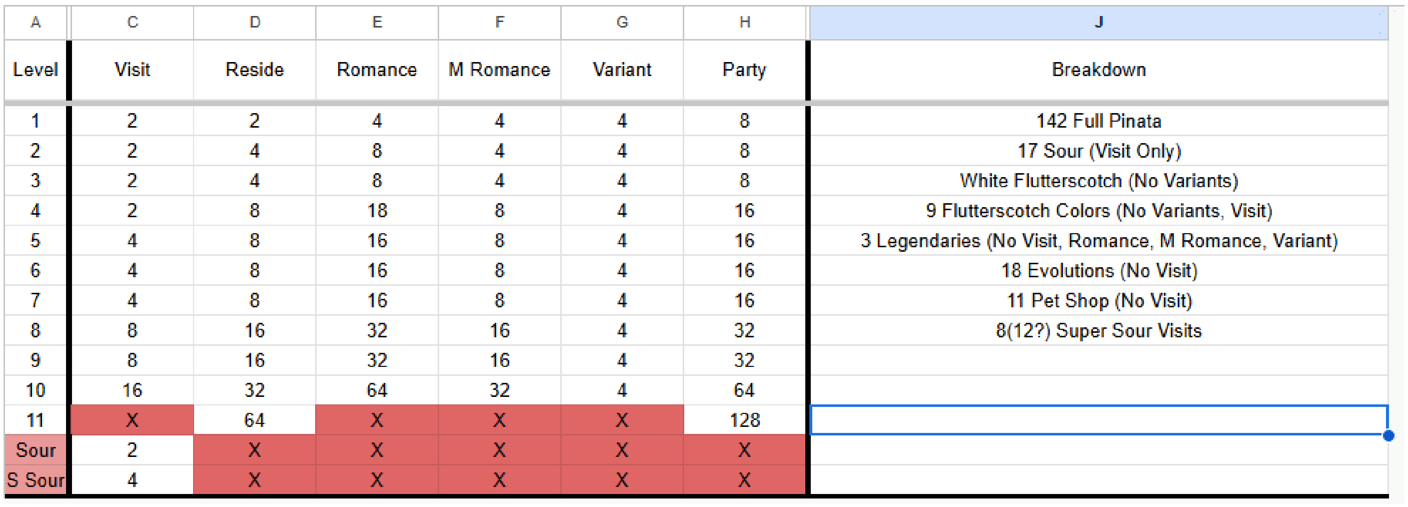
When you set the Animal Level it will automatically update the experience each task will give you expect variants, variants remain 4 experience no matter the level of the pinata.



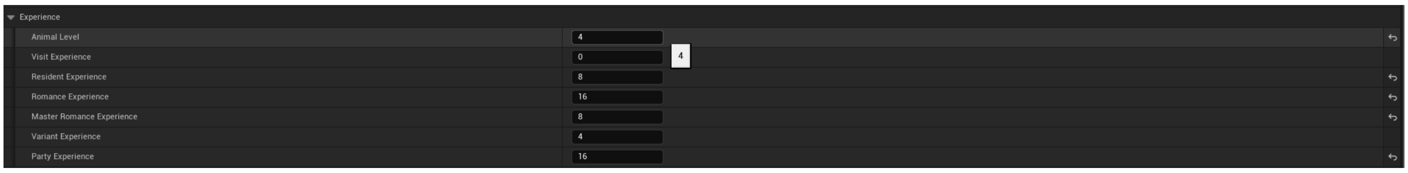
```
//Array for default visit experience based on pinata level
TArray<int> VisitExperienceArray = { 2, 2, 2, 2, 4, 4, 4, 8, 8, 16, 0 };

//Array for default Reside experience based on pinata level
TArray<int> ResideExperienceArray = { 2, 4, 4, 8, 8, 8, 8, 16, 16, 32, 64 };

//Array for default Romance experience based on pinata level
TArray<int> RomanceExperienceArray = { 4, 8, 8, 16, 16, 16, 16, 32, 32, 64, 0 };

//Array for default Master Romance experience based on pinata level
TArray<int> MasterRomanceExperienceArray = { 4, 4, 4, 8, 8, 8, 8, 16, 16, 32, 0 };

//Array for default Party experience based on pinata level
TArray<int> PartyExperienceArray = { 8, 8, 8, 16, 16, 16, 16, 32, 32, 64, 128 };
```

The values per level can be found in the C++ as arrays, they are 1 indexed so the first element is level one

These values I set are from Ici's research of the values in TIP but will likely need to be adjusted overtime for RTP



A	C	D	E	F	G	H	J
Level	Visit	Reside	Romance	M Romance	Variant	Party	Breakdown
1	2	2	4	4	4	8	142 Full Pinata
2	2	4	8	4	4	8	17 Sour (Visit Only)
3	2	4	8	4	4	8	White Flutterscotch (No Variants)
4	2	8	18	8	4	16	9 Flutterscotch Colors (No Variants, Visit)
5	4	8	16	8	4	16	3 Legendaries (No Visit, Romance, M Romance, Variant)
6	4	8	16	8	4	16	18 Evolutions (No Visit)
7	4	8	16	8	4	16	11 Pet Shop (No Visit)
8	8	16	32	16	4	32	8(12?) Super Sour Visits
9	8	16	32	16	4	32	
10	16	32	64	32	4	64	
11	X	64	X	X	X	128	
Sour	2	X	X	X	X	X	
S Sour	4	X	X	X	X	X	



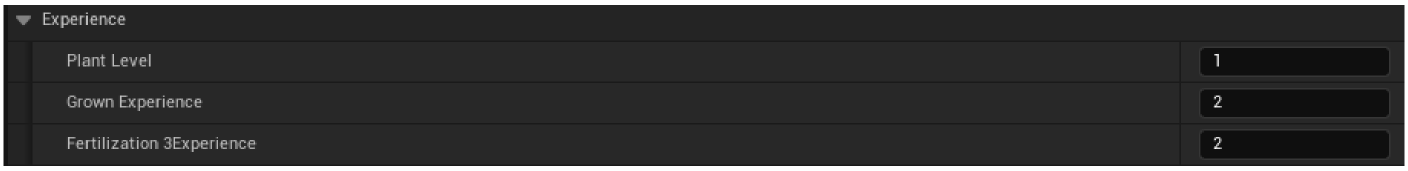
Experience

Animal Level	4	
Visit Experience	0	4
Resident Experience	8	
Romance Experience	16	
Master Romance Experience	8	
Variant Experience	4	
Party Experience	16	

Heres an example of setting the level to 4, I didnt adjust the lower values they automatically got updated. However after setting the level you can adjust the lower values freely

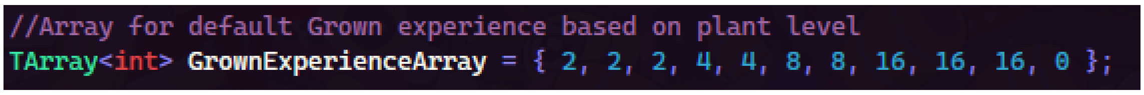
DataAssets

Plant Definition Data



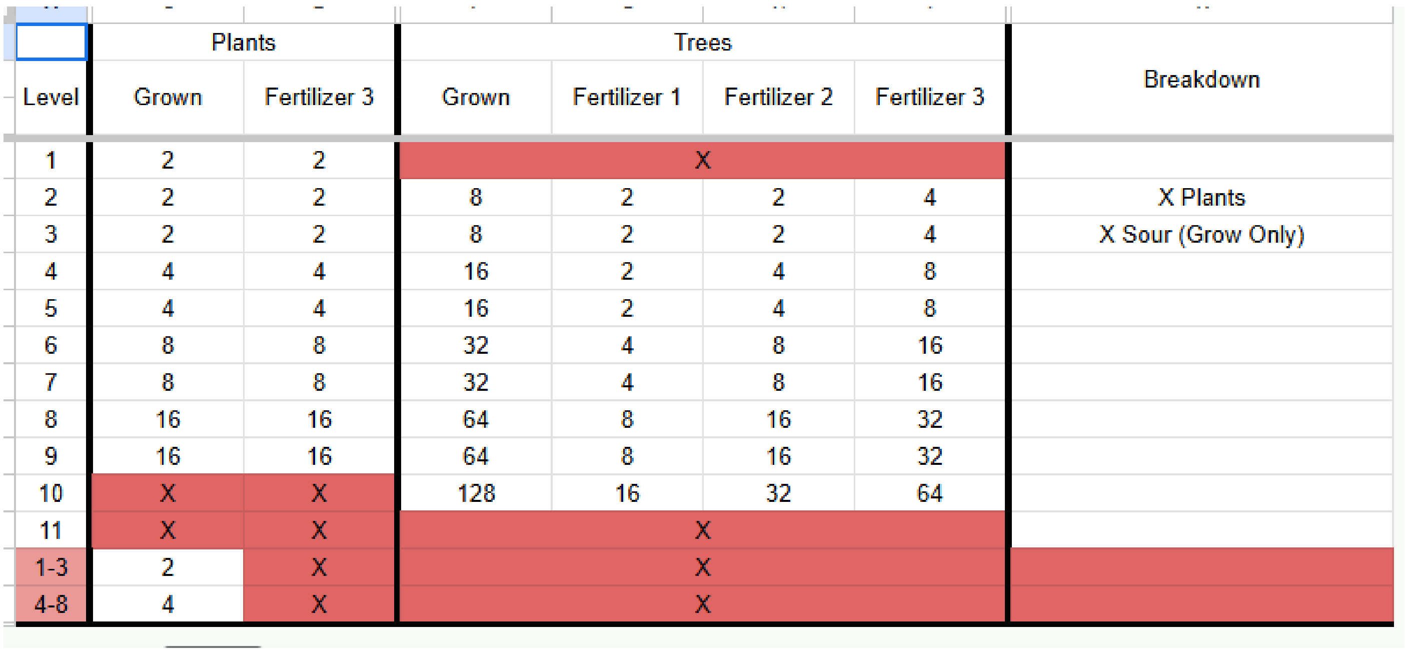
Experience	
Plant Level	1
Grown Experience	2
Fertilization 3Experience	2

The Plant experience is fairly straightforward the player has to fully grow a plant to get the growth experience and the fertilize experience is equal to the growth experience. But do note that the plants have to be fertilized 3 times to get the experience for fertilizing.



```
//Array for default Grown experience based on plant level
TArray<int> GrownExperienceArray = { 2, 2, 2, 4, 4, 8, 8, 16, 16, 16, 0 };
```

The Grown Experience is all that needs to be changed, the fertilization experience is automatically calculated from the grown experience. These values where also studied by Ici from TIP



	Plants		Trees				
Level	Grown	Fertilizer 3	Grown	Fertilizer 1	Fertilizer 2	Fertilizer 3	Breakdown
1	2	2	X				
2	2	2	8	2	2	4	X Plants
3	2	2	8	2	2	4	X Sour (Grow Only)
4	4	4	16	2	4	8	
5	4	4	16	2	4	8	
6	8	8	32	4	8	16	
7	8	8	32	4	8	16	
8	16	16	64	8	16	32	
9	16	16	64	8	16	32	
10	X	X	128	16	32	64	
11	X	X	X				
1-3	2	X	X				
4-8	4	X	X				

The tree data asset has special calculations for the Growth experience and fertilization experience, but in short it goes as follows:

Grown Experience is multiplied by 4 for trees

Fertilizer 3 is Grown Experience divided by 2

Fertilizer 2 is Fertilizer 3 divided by 2

Fertilizer 1 is Fertilizer 2 divided by 2 unless Fertilizer 2 is two in which case Fertilizer 1 is 2

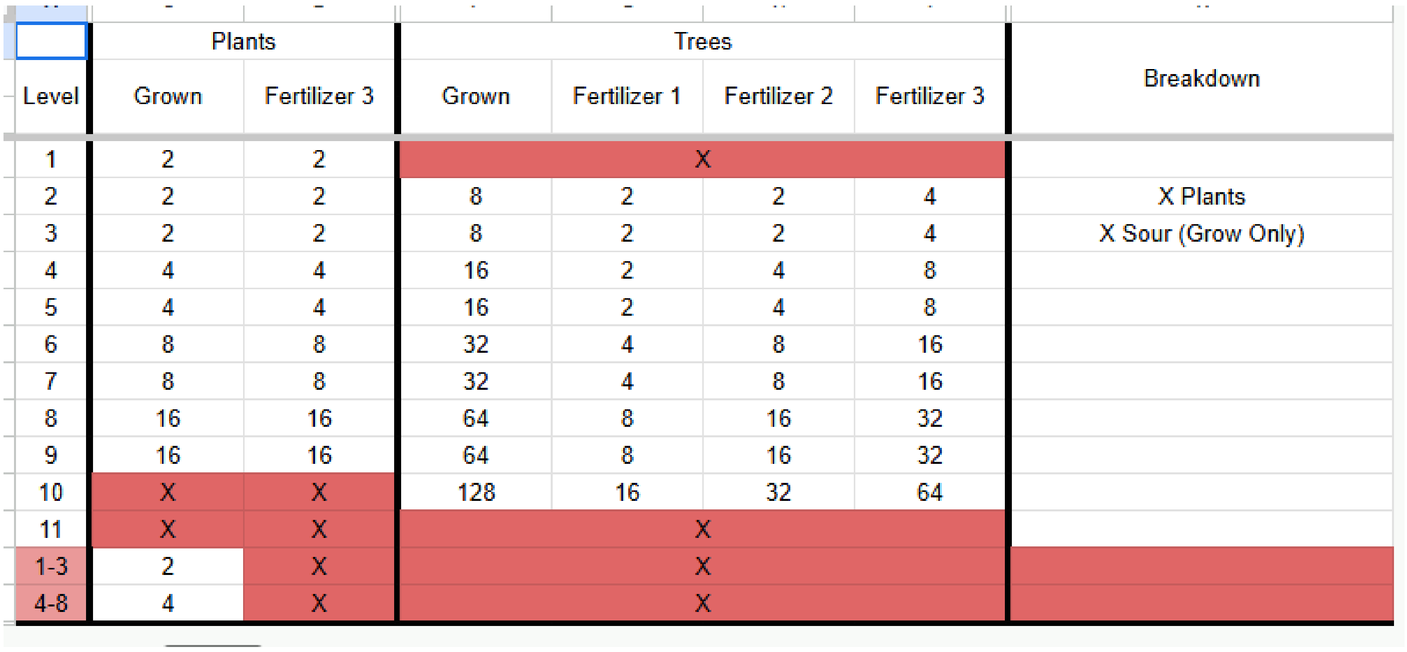
Tree Definition Data

Grown Experience is multiplied by 4 for trees

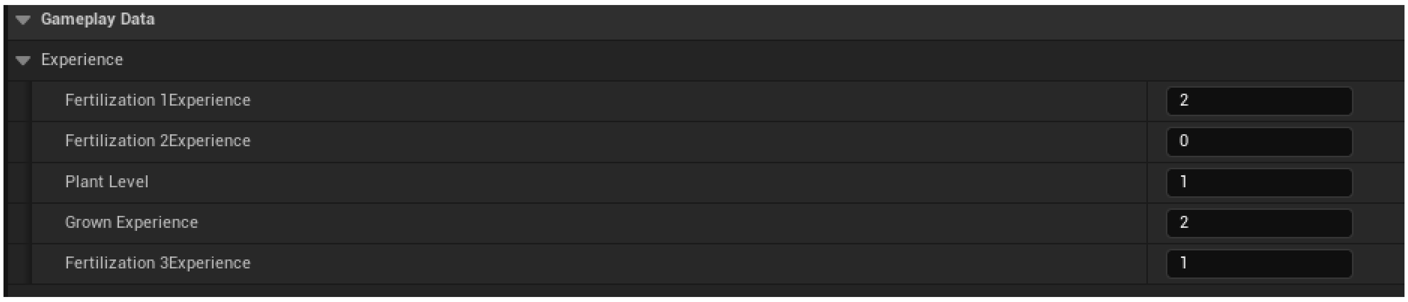
Fertilizer 3 is Grown Experience divided by 2

Fertilizer 2 is Fertilizer 3 divided by 2

Fertilizer 1 is Fertilizer 2 divided by 2 unless Fertilizer 2 is two in which case Fertilizer 1 is 2



	Plants		Trees				Breakdown
Level	Grown	Fertilizer 3	Grown	Fertilizer 1	Fertilizer 2	Fertilizer 3	
1	2	2	X				
2	2	2	8	2	2	4	X Plants
3	2	2	8	2	2	4	X Sour (Grow Only)
4	4	4	16	2	4	8	
5	4	4	16	2	4	8	
6	8	8	32	4	8	16	
7	8	8	32	4	8	16	
8	16	16	64	8	16	32	
9	16	16	64	8	16	32	
10	X	X	128	16	32	64	
11	X	X	X				
1-3	2	X	X				
4-8	4	X	X				

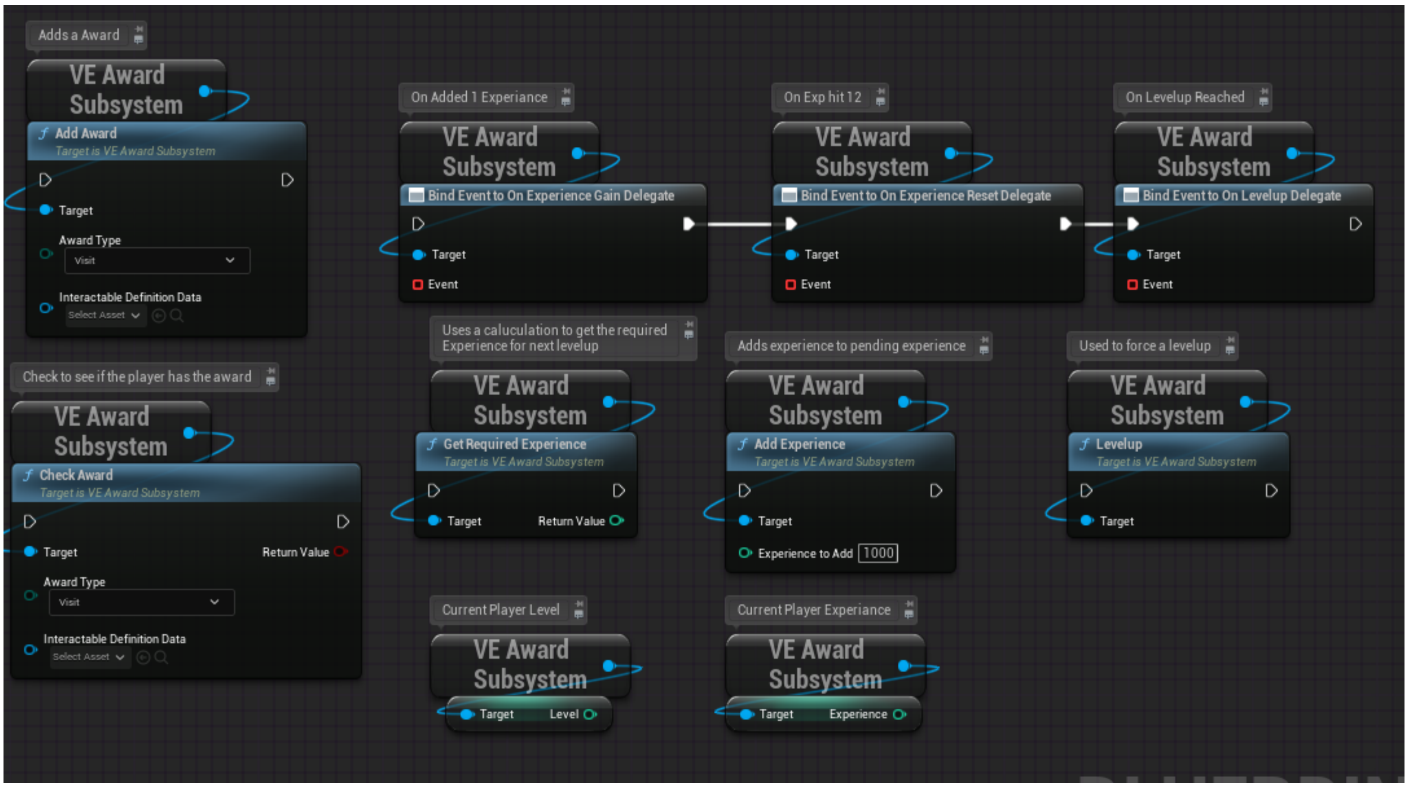


Gameplay Data	
Experience	
Fertilization 1Experience	2
Fertilization 2Experience	0
Plant Level	1
Grown Experience	2
Fertilization 3Experience	1

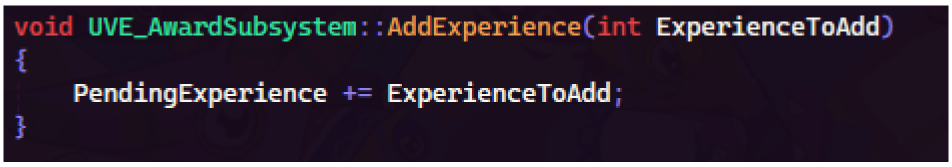
Award Subsytem

The controller for the experience and levelups, also has award storage

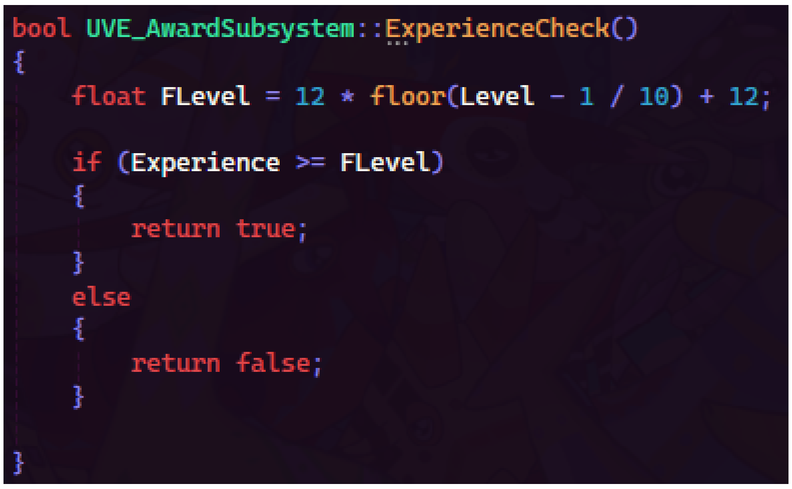
Blueprint functions



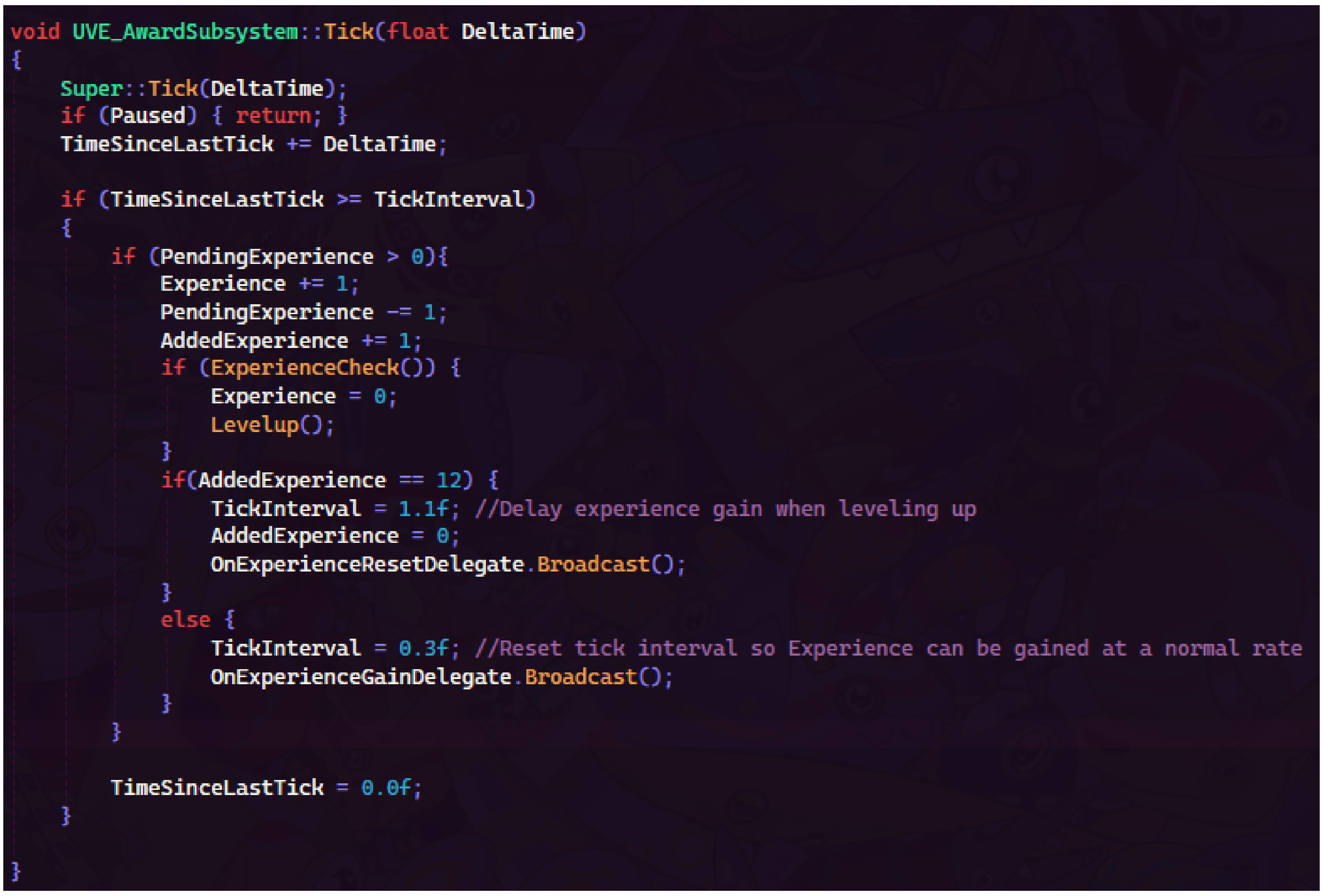
Adding Experience



```cpp
void UVE_AwardSubsystem::AddExperience(int ExperienceToAdd)
{
    PendingExperience += ExperienceToAdd;
}
```



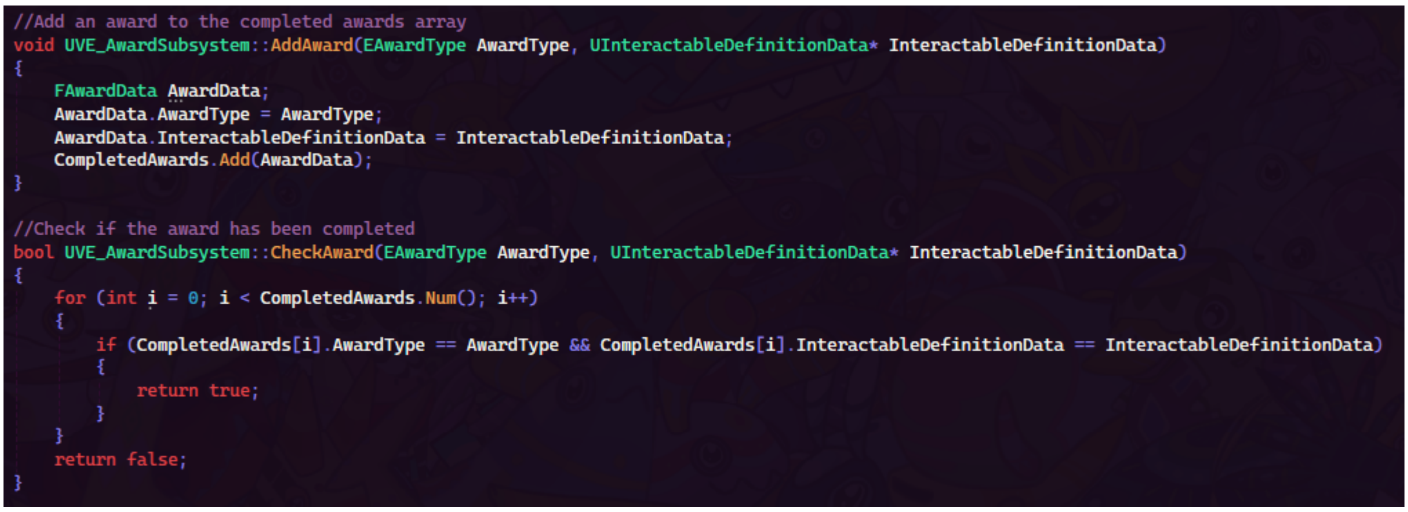
```cpp
bool UVE_AwardSubsystem::ExperienceCheck()
{
    float FLevel = 12 * floor(Level - 1 / 10) + 12;

    if (Experience >= FLevel)
    {
        return true;
    }
    else
    {
        return false;
    }

}
```



```cpp
void UVE_AwardSubsystem::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
    if (Paused) { return; }
    TimeSinceLastTick += DeltaTime;

    if (TimeSinceLastTick >= TickInterval)
    {
        if (PendingExperience > 0){
            Experience += 1;
            PendingExperience -= 1;
            AddedExperience += 1;
            if (ExperienceCheck()) {
                Experience = 0;
                Levelup();
            }
            if(AddedExperience == 12) {
                TickInterval = 1.1f; //Delay experience gain when leveling up
                AddedExperience = 0;
                OnExperienceResetDelegate.Broadcast();
            }
            else {
                TickInterval = 0.3f; //Reset tick interval so Experience can be gained at a normal rate
                OnExperienceGainDelegate.Broadcast();
            }
        }

        TimeSinceLastTick = 0.0f;
    }

}
```

The add experience function adds experience to the pending experience variable, the tick then moves one point at a time to the experience variable leveling up along the way, every 12 the clock will need to reset

Adding/Checking Awards



```cpp
//Add an award to the completed awards array
void UVE_AwardSubsystem::AddAward(EAwardType AwardType, UInteractableDefinitionData* InteractableDefinitionData)
{
    FAwardData AwardData;
    AwardData.AwardType = AwardType;
    AwardData.InteractableDefinitionData = InteractableDefinitionData;
    CompletedAwards.Add(AwardData);
}

//Check if the award has been completed
bool UVE_AwardSubsystem::CheckAward(EAwardType AwardType, UInteractableDefinitionData* InteractableDefinitionData)
{
    for (int i = 0; i < CompletedAwards.Num(); i++)
    {
        if (CompletedAwards[i].AwardType == AwardType && CompletedAwards[i].InteractableDefinitionData == InteractableDefinitionData)
        {
            return true;
        }
    }
    return false;
}
```