Saving and Loading

The logic behind how the game saves and loads

- Saving Templates
  - FVE_BaseSaveStats
  - FVE_PlayerStats and FVE_StarLeafView

Saving Templates

The template structs that hold the saving data for saved item types

FVE_BaseSaveStats

The base struct type that makes up all of the saving structs. It holds the transform (location, rotation, and scale), Gameplay Tags (labels for portraying easily accessible identifying information), and the class type of the object being saved.

```
USTRUCT(BlueprintType)
struct FVE_BaseSaveStats {

    GENERATED_BODY()

    //The transform of the object at the moment of saving.
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FTransform ObjectTransform;

    //The gameplay tags that the object has.
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FGameplayTagContainer gameplayTags;

    //The class of the saved object. Only used for loading objects that are not pooled.
    UPROPERTY(meta = (MustImplement = "Saveable"))
    TSubclassOf<UObject> objectClass;
};
```

FVE_PlayerStats and FVE_StarLeafView

The PlayerStats are the first major branch-off point of the saving types, as the things the player needs to save are wildly different from your average items. FVE_StarLeafView holds the data for if a shop item is unlocked and if it should display the cloud effect for new items.

FVE_PlayerStats adds the following variables:

FString profileDisplayName: The name of the player profile, usually set to the profile name of the platform running the game (Steam). This string is used to determine which directory to look for when saving to and loading from garden files.

int64 coins: The amount of money the player has. This number persists through all gardens, so it's saved here rather than to the garden itself.

int32 level: The player's current level. It's only the level; the amount of experience contributing to a level up is stored in a separate integer.

int32 experience: The amount of experience that the player has so far. This is only the experience; the player's level is stored as a separate integer.

TMap<int32, bool> unlockedAnimals: The animals whose appear requirements have been satisfied. The key is the definition ID of the animal (like the species), and the value is whether or not it has been unlocked. If the definition ID is not in the map, then it is assumed to be locked by default.

```cpp
USTRUCT(BlueprintType)
struct FVE_StarLeafView
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    bool isUnlocked = false;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    bool cloudVisable = true;
};

USTRUCT(BlueprintType)
struct FVE_PlayerStats : public FVE_BaseSaveStats {

    GENERATED_BODY()

    //The name of the player profile.
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FString profileDisplayName;

    //The amount of money the player has.
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int64 coins;

    //The player's current level.
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 level;

    //The amount of experience that the player has so far.
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 experience;

    //A map of all unlocked animals. If animal definition isn't present, then the animal is assumed to be locked.
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TMap<int32, bool> unlockedAnimals;

    //Which leaves are locked and unlocked
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TMap<int32, FVE_StarLeafView> leafCloudVisability;
};
```