BLUEPRINTS TO C++

UNREAL ENGINE 4 - C++ PROGRAMMING GUIDE

EPISODE 7

TMAP BASICS

OUTLINE

1. TMap Function Blueprint Comparison
2. TMap Iteration Types
3. TMap Functions Recap
4. Tip of the Day

TMAP BLUEPRINT/C++ FUNCTION COMPARISON

• Add (Blueprint)	= Map.Add(Key,Value);
• Length (Blueprint)	= Map.Num();
• Contains Item (Blueprint)	= Map.Contains(Key);
• Find (Blueprint)	= Map.Find(Key);
• Keys (Blueprint)	= Map.GetKeys(KeysArray);
• Values (Blueprint)	= Map. GenerateValueArray(ValuesArray);
• Remove Item (Blueprint)	= Map.Remove(Key);
• Clear (Blueprint)	= Map.Empty();

STANDARD FOR LOOP ONLY WITH KEY ARRAY

```
TArray<int32> Keys;
Int32 Num = Map.GetKeys(Keys);
….
for(int32 i=0; i < Num; i++)
{
	const int32 Key = Keys[i];
	FVector& Vec = Map[Key];
}
```

TMAP RANGE BASE FOR EACH LOOP

```
TMap<int32,FVector> Map;
…
for(const TPair<int32, FVector>& Kvp : Map)
{
    const int32 Key = Kvp.Key;
    const FVector& = Kvp.Value;
}
for(const auto& Kvp : Map)
{
    const int32 Key = Kvp.Key;
    const FVector& = Kvp.Value;
}
```

TMAP ITERATORS

```
TMap<int32,FVector> Map;
….
for (auto It = Map.CreateIterator(); It; ++It)
{
    int32 MapKey = It.Key();
    FVector& Vec = It.Value();
}
for (auto It = Map.CreateConstIterator(); It; ++It)
{
    const int32 MapKey = It.Key();
    const FVector& Vec = It.Value();
}
```

TMAP IMPORTANT FUNCTIONS RECAP

1. Add – Adds a new Value based on Key
2. Append – Appends another map to the map
3. GetKeys – Fills a passed Array with the keys from the map
4. GenerateValueArray – Fills a passed Array with the values from the map
5. Contains – Checks to see if map contains the key
6. Find – Finds the pointer to a value in the map based on key
7. Num – returns the number of elements
8. Empty – Clears the whole set
9. Remove – Removes an entry from the map based on key

TIP OF THE DAY – FIND WITH POINTER TYPE VALUES

```
TMap<int32,AConeActor*> ConeActorMap;

const int32 Key = 1;

...

        AConeActor** ConeActorPtr = ConeActorMap.Find(Key);

        AConeActor* ConeActor = *ConeActorPtr;

Or

        AConeActor* ConeActor = *ConeActorMap.Find(Key);

…

If(ConeActor)

{

  …

}
```

Find always returns a pointer to the value. In case of a Pointer class like AConeActor* it returns a pointer to a pointer. So to access the pointer you need to dereference the pointer of AConeActor**

THANK YOU FOR WATCHING

IF YOU WANT TO GET NOTIFIED WHEN NEW VIDEOS ARE COMING OUT

THEN PLEASE SUBSCRIBE TO THE CHANNEL