GED Course Project

Members

Devon - Student number: 100861866
Contributions: Batching, Flyweight, document management
Percent: 33%

Daniel - Student number: 100820942
Contributions: DLL,
Percent: 40%

Jacob - Student number: 100830985
Contributions: Object Pooling, document management
Percent: 27%

Use of third-party resources

- FMOD was used to assist in managing audio assets. This is justified by the project being a racing game, which necessitates the use of adaptive audio, which is something that FMOD excels at.
- Audio assets, such as background music and menu sound effects, were created by non-GED members of the GDW team.
- The initial framework of the audio management code was created by Tom, a member of the GDW team, as they have the most experience with FMOD. The justification is that FMOD can be confusing when you first start, and having something to work off of helps a ton.

Game Design Pattern improvements from the previous assignment /20
improvement / modifications done to assignment 1 deliverables:
○ Include visual evidence (screenshots) that supports the improvements.
○ Explain why the improvements were made in order to best serve the game or interactive media scenario submitted as part of the course project.

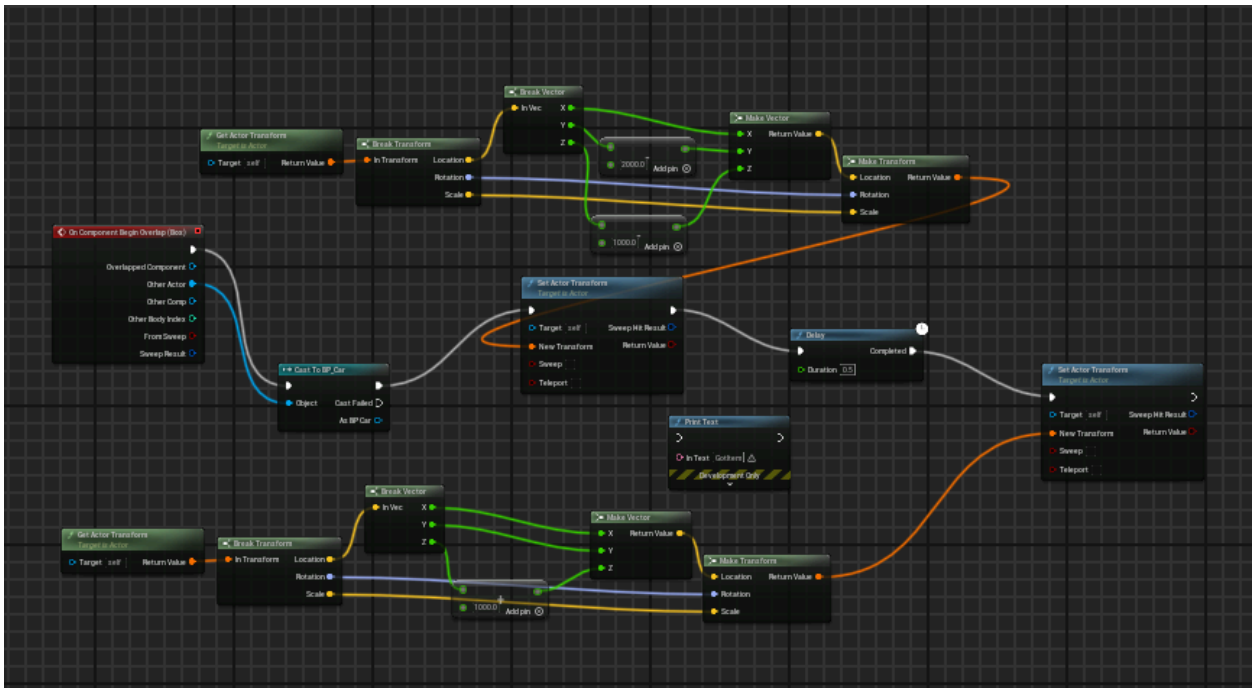
We most likely did not change them.

Optimization Design Patterns /30
3 design patterns required
○ Example implementations: Flyweight, object pool, batching, or dirty flag.”
○ Working game design patterns (live demonstration).
○ Description of how the design patterns were implemented supported with a flowchart and in-game footage (report).
○ Explanation indicating why the implementation was best suited for the course Project.

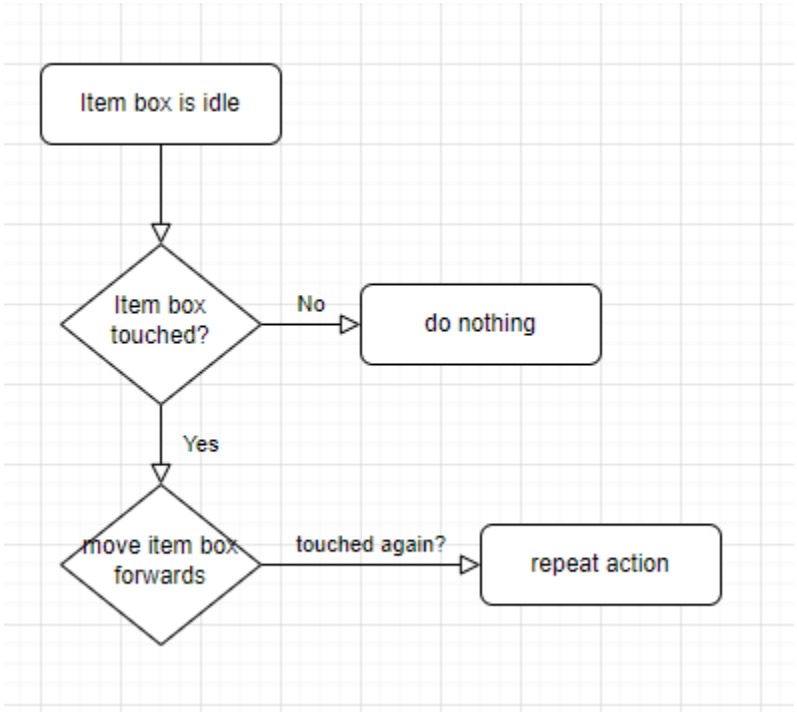
Object Pooling

There are 5 item boxes that are in a pool. When the player touches an item box, it goes into the ground. The item box will then reappear in front of the player. This has been chosen to save on resources and make managing the item boxes easier, since there will only be 5 of them.

Code:



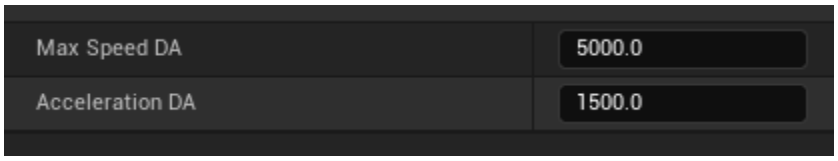
flowchart:



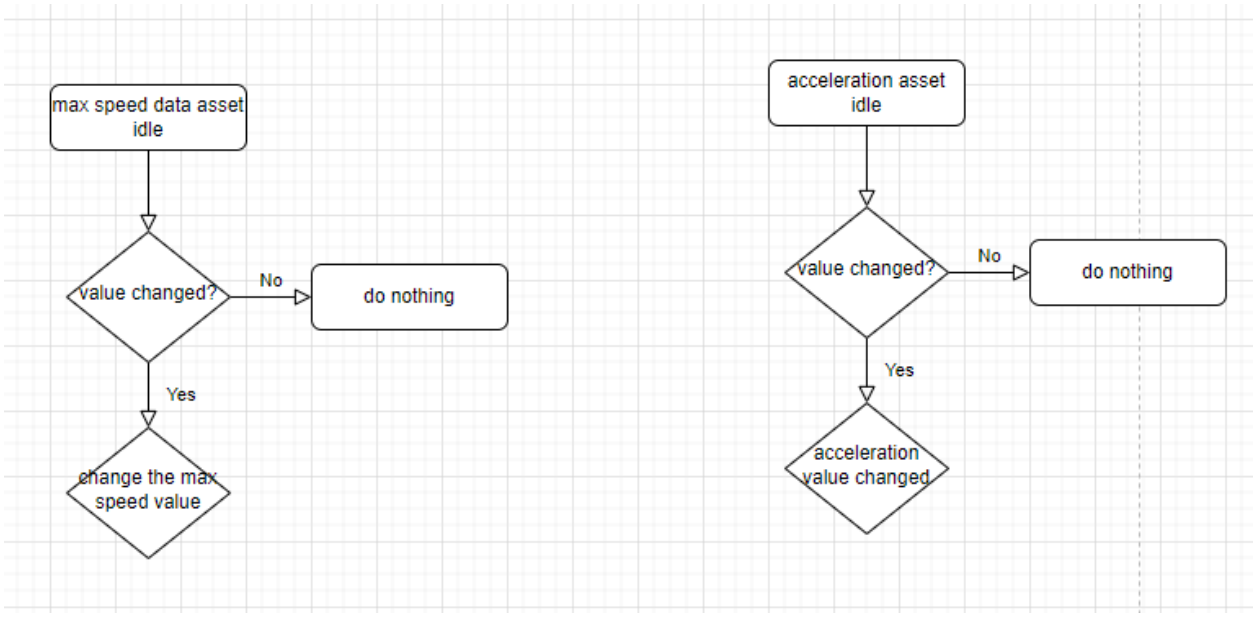
Flyweight

Data assets have been used to set the max speed and acceleration values for the vehicle. This will aid in optimizing the game by having several vehicles - AI or human - share these values.

Data Asset



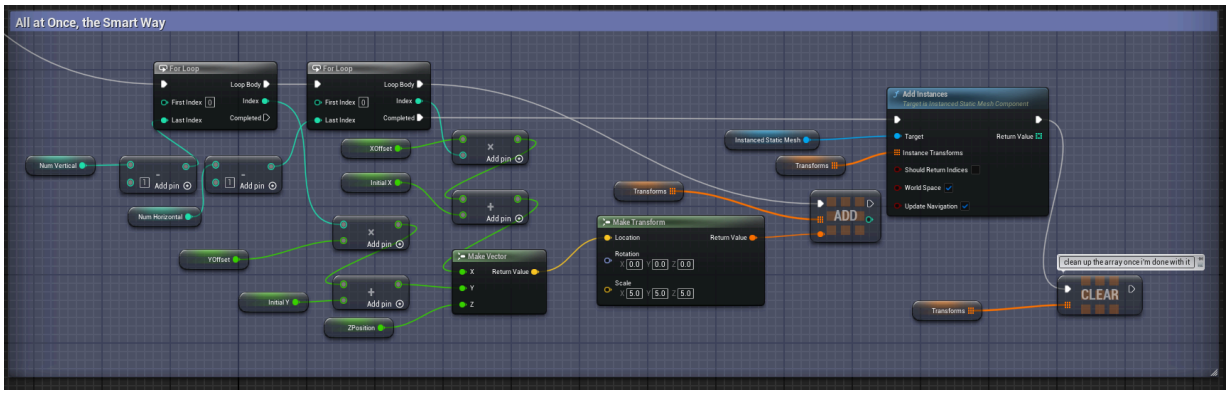
Flowchart:



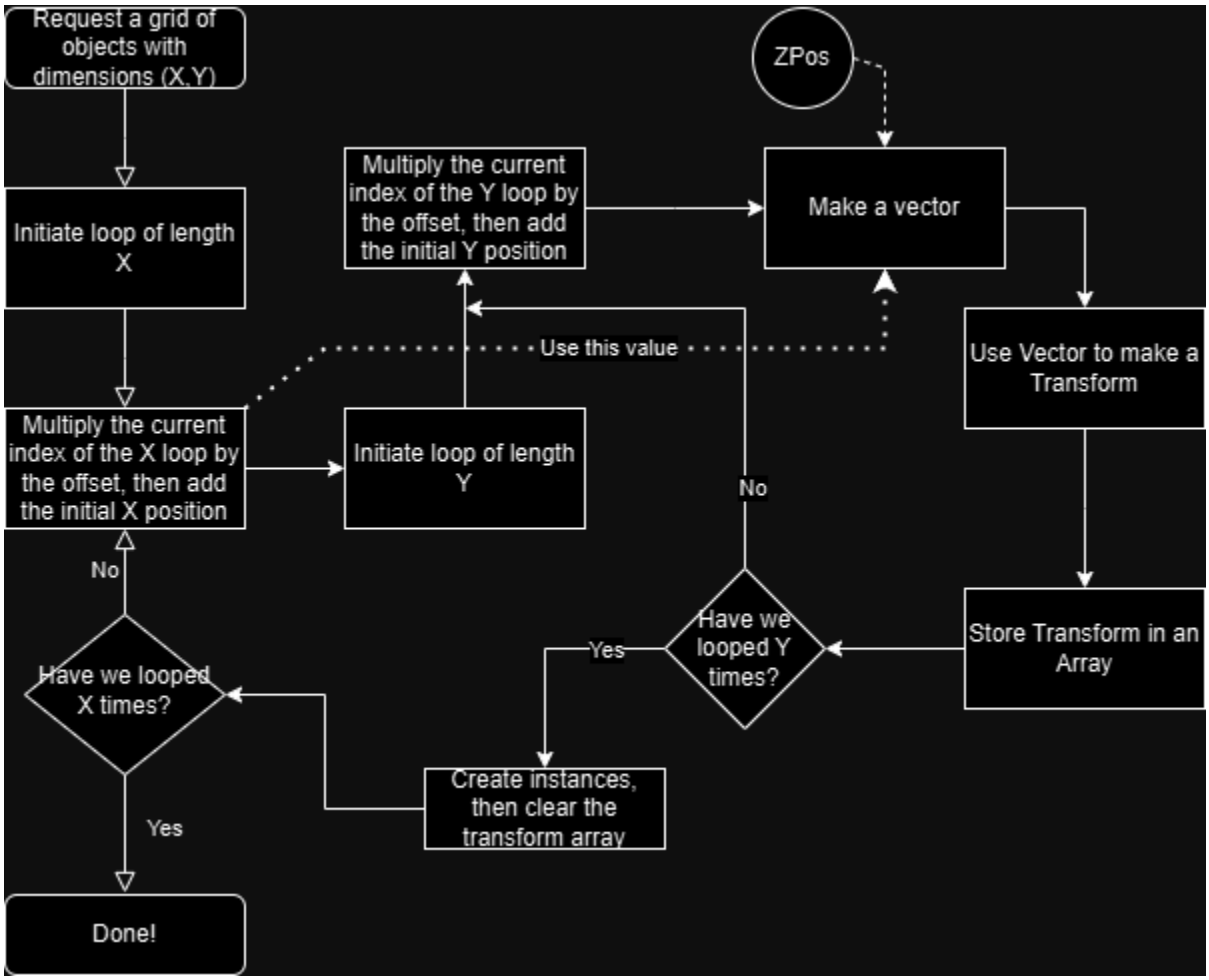
Batching

Batching was implemented through the use of For loops and AddInstances. This implementation was best suited for the course project because several solar panels were already present, and it was only natural to turn their creation into one call.

Code:



Flowchart:

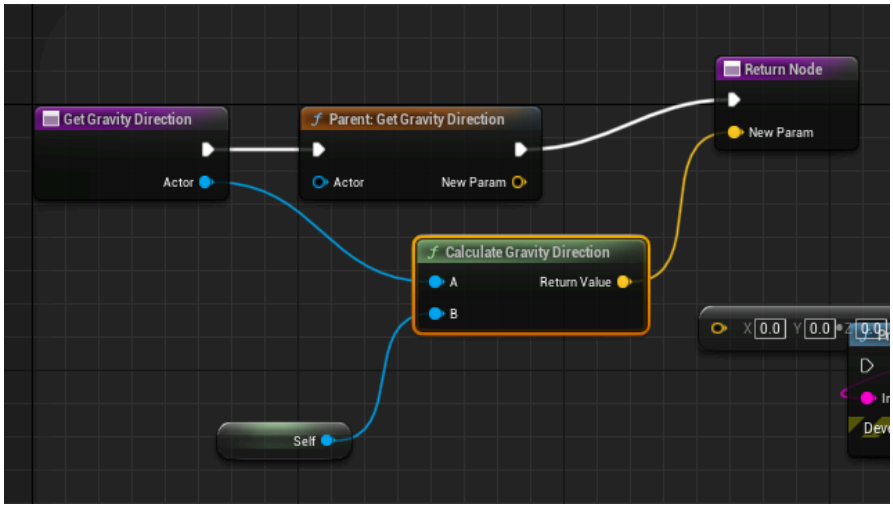


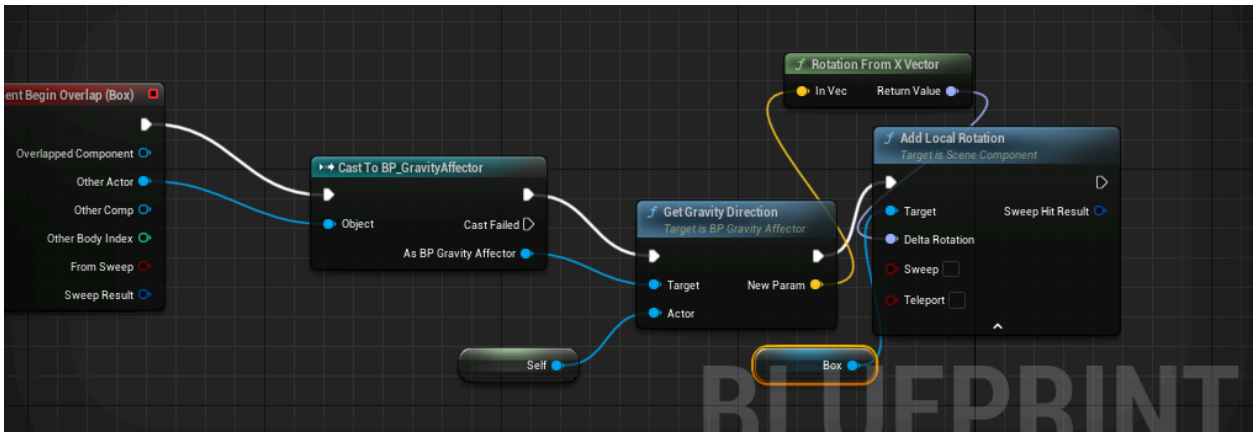
Plugin

The plugin is a way to grab vectors from a surface to the player's own upwards vector, those two vectors are then used to determine which direction the angle of the player should be.
Code:

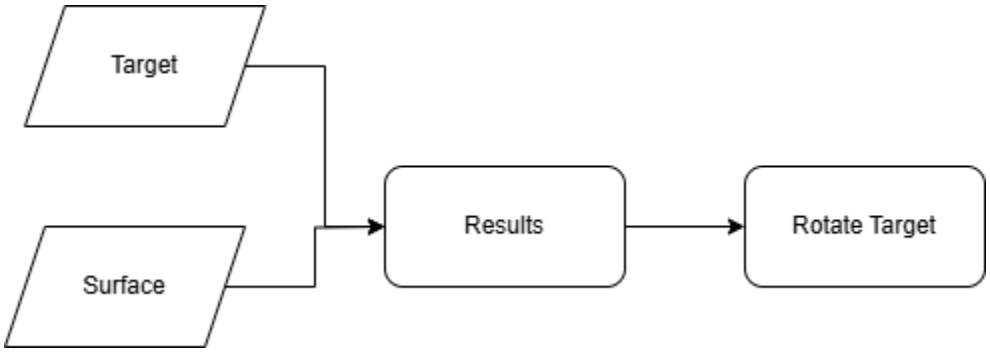
```h
#pragma once

#include "Kismet/BlueprintFunctionLibrary.h"
#include "GravityBPLibrary.generated.h"


UCLASS()
class UGravityBPLibrary : public UBlueprintFunctionLibrary
{
    GENERATED_BODY()

    UFUNCTION(BlueprintPure)
    static FVector CalculateGravityDirection(AActor* a, AActor* b);

};
```

```cpp

#include "GravityBPLibrary.h"
#include "Gravity.h"

/*
UGravityBPLibrary::UGravityBPLibrary(const FObjectInitializer& ObjectInitializer)
: Super(ObjectInitializer)
{

}
*/

FVector UGravityBPLibrary::CalculateGravityDirection(AActor* a, AActor* b)
{
    FVector c;
    c = (b->GetActorLocation() - a->GetActorLocation()).GetSafeNormal();

    return c;


}
```





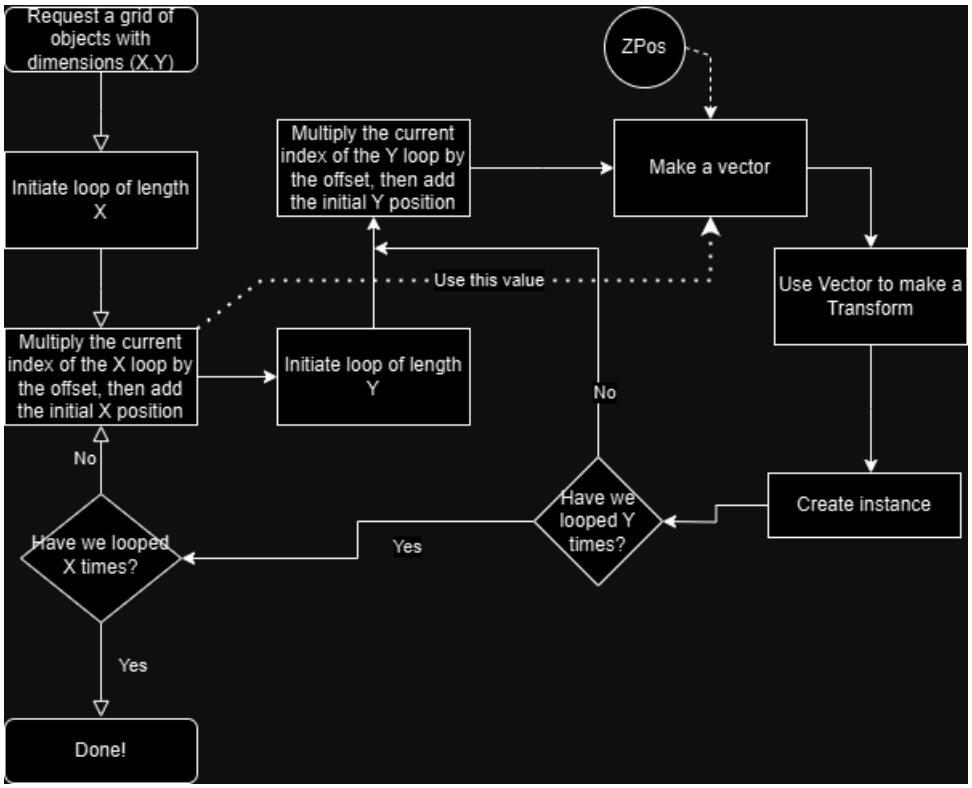
Flowchart:



Performance Profiling

Performance improvement was implemented by replacing the AddInstance function with AddInstances, reducing the number of function calls by 9994 (when comparing the Unoptimized version using AddInstance vs the Optimized version using AddInstances).

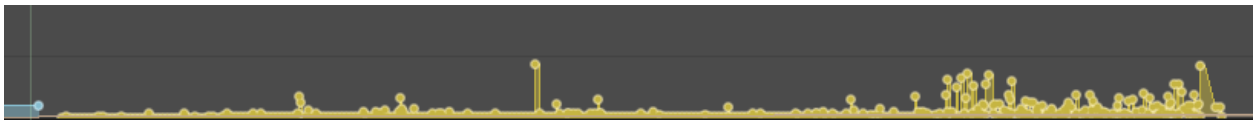
Flowchart (Unoptimized Solar Panel Creation):



Blue: FMemory::Trim
Yellow: AddInstance
Green: AddInstances
Pink: Instance

Unoptimized:

(Close up of end)

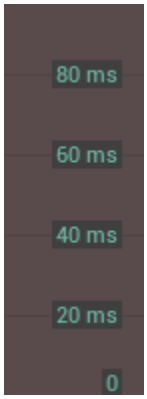


Count + Time:

With Batching:

Count + Time:

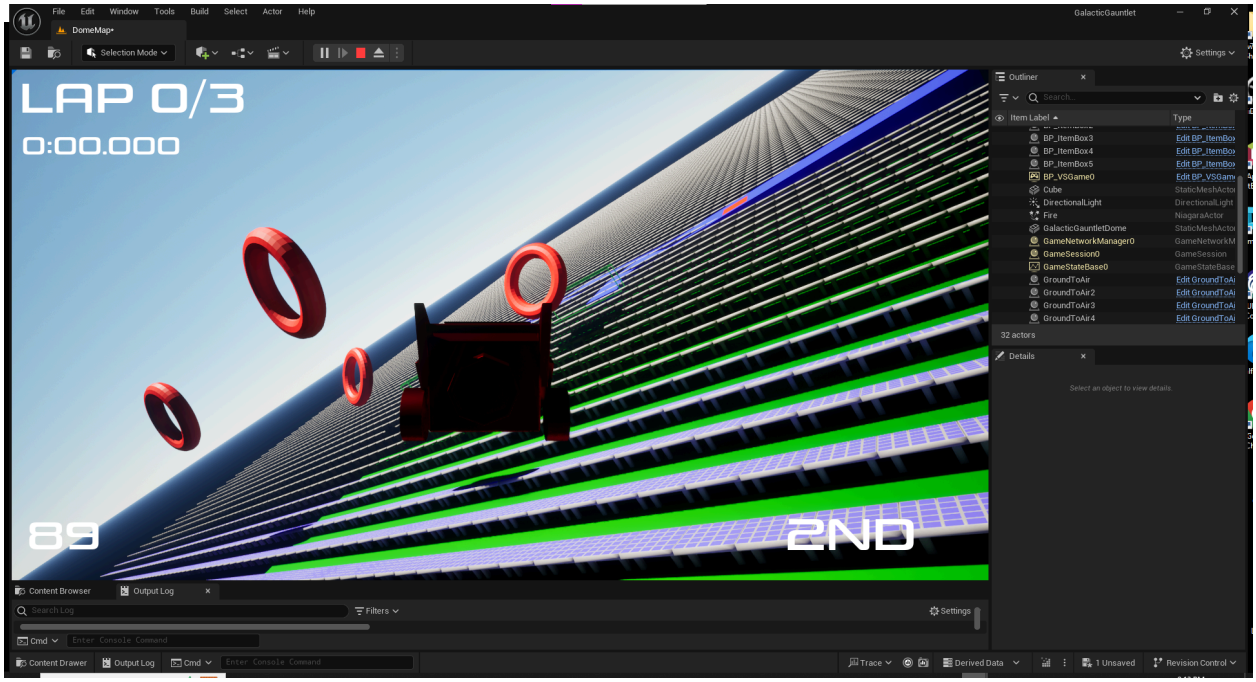
Scale (same for both):



Results of testing:

	How long does it take in seconds to spawn 10,000 solar panels if you spawn them:	
Trial #	One at a Time	All at Once
1	3.989	0.601
2	3.862	0.571
3	3.848	0.576
4	3.836	0.555
5	3.754	0.579
Average time (seconds)	3.8578	0.5764
Based on the formula below, it is 85.0588% faster to spawn all 10,000 solar panels at once vs one at a time.		

$$\text{Percentage Increase} = \frac{\text{Final Value} - \text{Starting Value}}{|\text{Starting Value}|} \times 100$$


File Edit Window Tools Build Select Actor Help
GalacticGauntlet
DomeMap*
Selection Mode
Settings
LAP 0/3
0:00.000
89
2ND
Outliner
Item Label
Type
BP_ItemBox3
BP_ItemBox4
BP_ItemBox5
BP_VSGame0
Cube
DirectionalLight
Fire
GalacticGauntletDome
GameNetworkManager0
GameSession0
GameStateBase0
GroundToAir
GroundToAir2
GroundToAir3
GroundToAir4
32 actors
Details
Select an object to view details
Content Browser
Output Log
Search Log
Filters
Settings
Cmd
Enter Console Command
Content Drawer
Output Log
Cmd
Enter Console Command
Trace
Derived Data
1 Unsaved
Revision Control