Task:

Codebase: [link to the project](https://drive.google.com/file/d/1SISamCYwjThYxGCFkwd6QHbicJwbe5X4/view?usp=sharing)

**1.** Setup input for walking, jumping, aiming and shooting the grappling hook in BP\_Character.

**2.** Implement a grappling hook mechanic in the form of a component:

* The player shoots a hook in a direction derived from their camera.
* Hook hits a target, the player starts being moved towards the grapple point.
* When the player reaches the grapple point, the target actor has its execution activated
* The player can let the hook off at any point in the interaction. When they do, execution should be deactivated and the rope should reset instantaneously.

Do not worry about the visual aspect of the rope or the hook’s ballistics. Make sure that it is clear where the rope starts and where the grappling hook hits.

**3.** Implement a uniform way of triggering custom execution upon a character reaching the grapple point of an actor.

* BP\_Carousel
* Execution activation should invoke the BeginRotating method.
* Execution deactivation should invoke the StopRotating method.
* BP\_Tower
* Execution activation should make the tower fall over. The character should be launched away from the tower and have the gravity scale decreased for several seconds.
* Execution deactivation does nothing.
* ANativeActor
* Execution activation should invoke the ANativeActor::Report() method.
* Execution deactivation does nothing.
* Note that this actor will deliberately not work if it is subclassed in a blueprint. It can only be derived from in C++.

You can subclass these types, but do not change their base definitions and declarations. The grappling hook component should be able to invoke custom execution without knowing explicit types and logic of the targets. Make it as easy as possible to expand this functionality to other actor types without the necessity to modify the grappling hook logic.

**4.** Prepare a simple level that enables derivatives of each of the actor types listed above to be tested.

**5.** Create a data asset type with two member variables - gravity scale and duration. Adapt your BP\_Tower's descendant implementation to infer the effect's scale and duration from the data asset referenced as an instance-editable member variable. Prepare three level-instances, each with different configuration data asset sets.

**6.** Make sure the solution behaves properly with two players in a split-screen mode.