# Combat system

Typical combat use case/scenario

*“The player is engaging an enemy. They’re being chased as they fly around the field. Suddenly, though, they release their thrusters, whip around and unleash a beam of energy that hits the enemy, knocks them away and depletes their health.”*

Already we can tell that the combat system is going to involve several other components of the game. Let’s make a list of the different components involved here.

* Player input
* Entity System/Actions/attacks
* Collision detection<
* Health States<
* Graphics/animation/particle effects<
* Ai<

playerInput->Entity/attack->collision->ai->healthStates->graphics

AttackAction Class

CollisionObserver Class

Basic Ai Input component class

StatusComponent

This all needs to interact with some kind of game event system.

EX of an event system flow for this

CollisionSystem is notified of collision event (Player,BeamFixture,Enemy,bodyFixture)

(possibly contain this stuff in a collisionevent struct)

This is placed in a queue and is resolved by the end of the update loop

CollisionSystem sends out an event notification to any who gives a shit for **both** Objects

In the case of the player (USED\_ATTACKBEAM\_EVENT,EntityA);

In the case of the enemy (HITBY\_ATTACKBEAM\_EVENT,EntityB);

And that is pretty much how things will go. I will possibly include within each entity data file a list of attacks it is capable to using as well as including in

## Attack Action Class

Attack action inherits from action. So it has an **execute(Actor entity)** function. The changes will be that each attack action has an attack fixture that it associates with the attack that it produces. Each Actor will have a list of attacks/skills. When each actor entity is created, if they have attacks/skills they will be set by the spawner in the form of a struct for containing necessary data possibly labeled **“attackStruct”.** The attack struct will have each attribute/data value for each attack neatly packed in.Attacks and skills may go hand in hand. Say for example, the “Joust attack,” is technically two actions. Boost and the Ram. Say an entity mostly moves using boost, (but boost has a cooldown after using it in rapid succession. But that’s beside the point). Boost would work with ram, because as they boost forward, they would also use ram and knock into their opponent. Thereby forming an attack using two actions.

Each input component has a list of actions based off of the entity that owns it

Each input component/controller has a base list of actions that every entity can use: accelerate,turn,etc