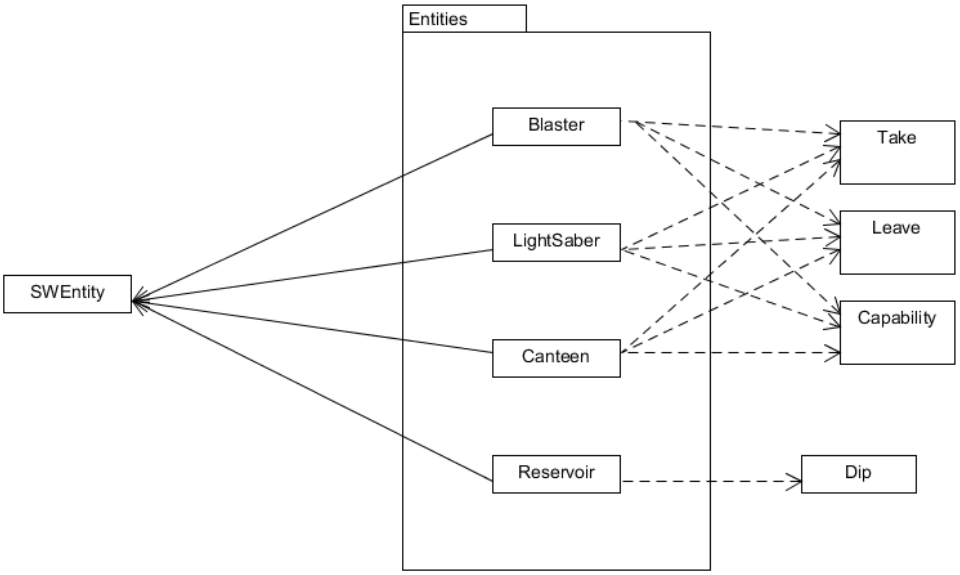
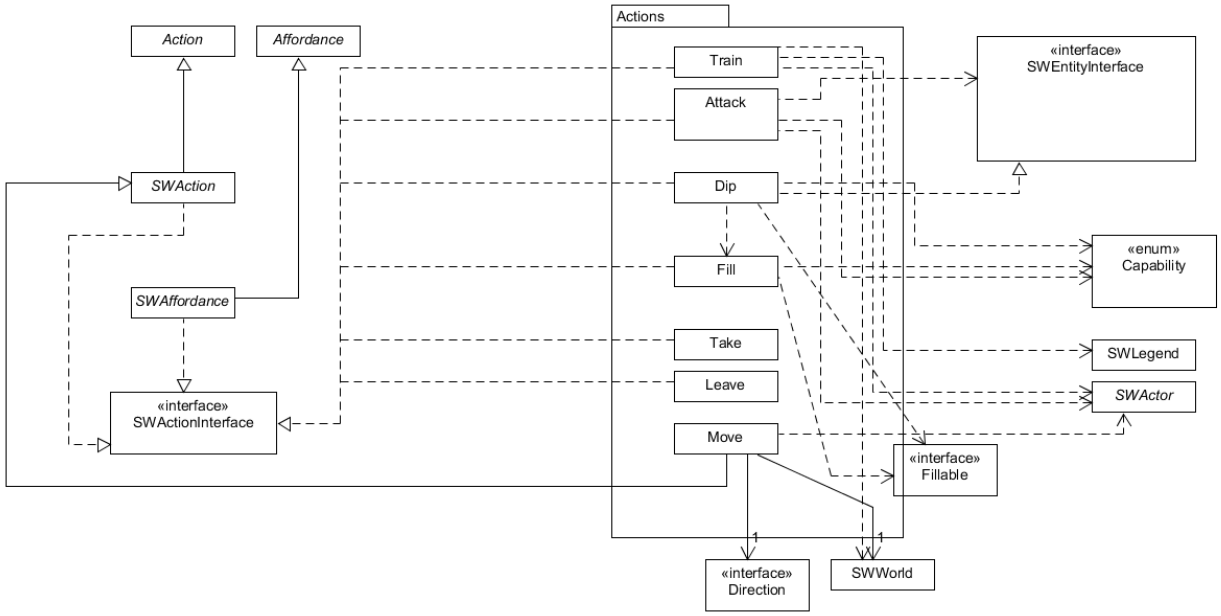
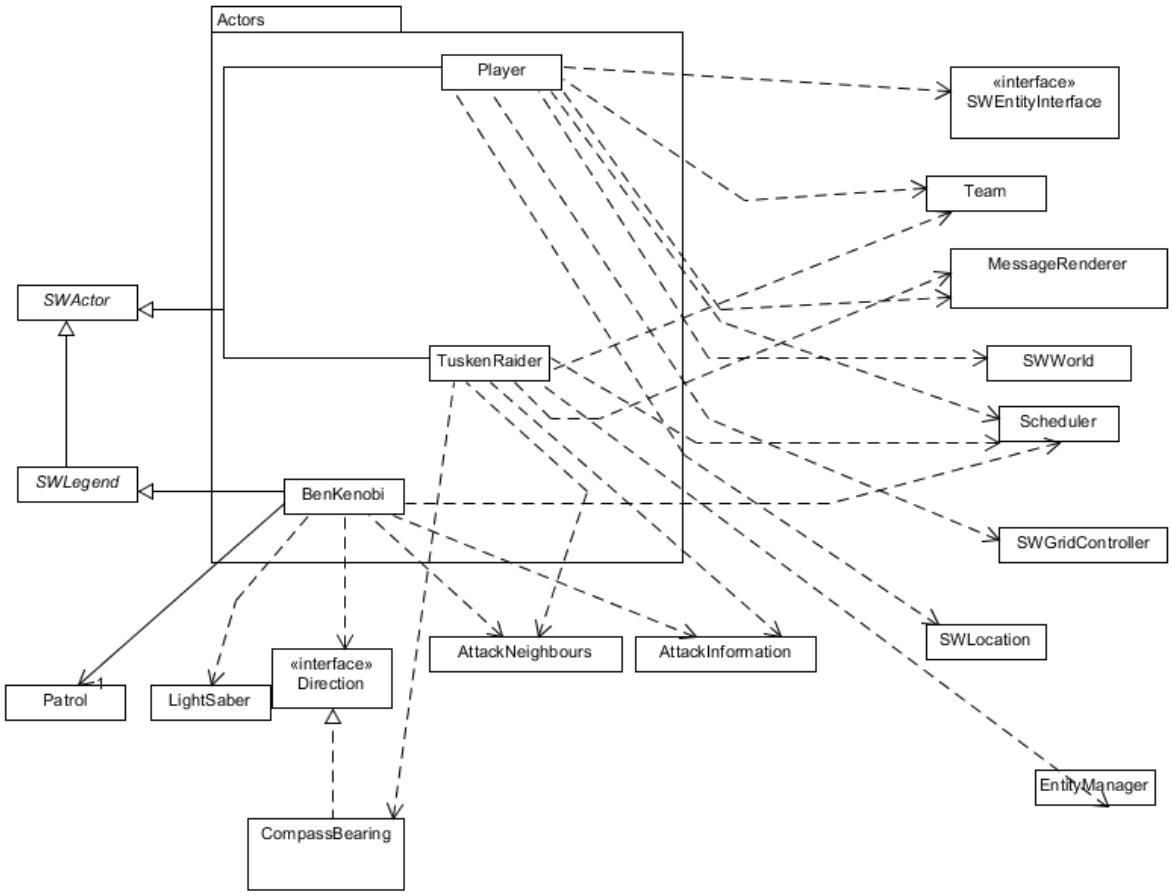
1. Leave Affordance:





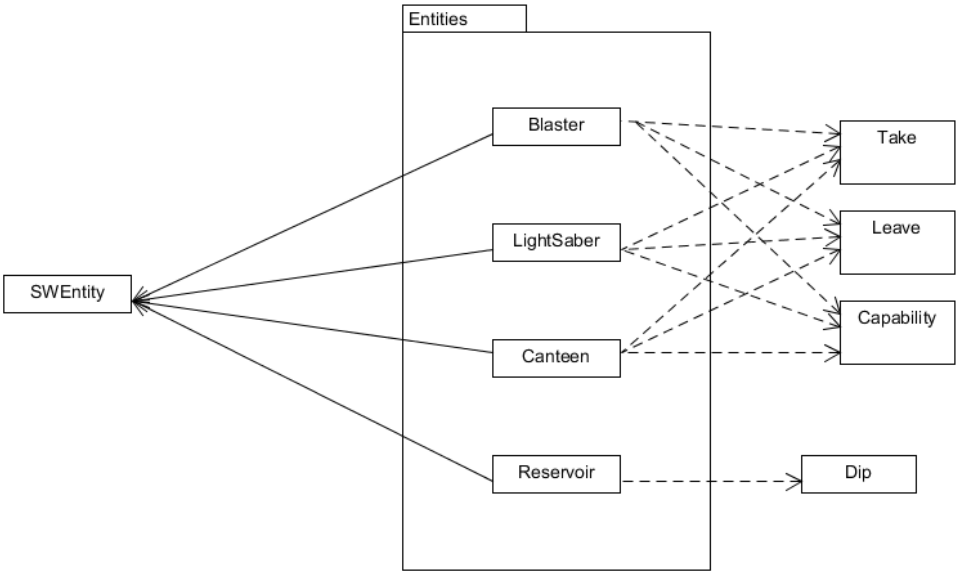
- Leave is a new class created in A class called Leave in starwars.actions. It extends the SWAffordance class. The class contains the methods “canDo”, “act”. The method canDo checks if the actor is carrying an entity. The method act checks if the target is an Entity. If so, it assigns it to a variable. Then we call the method “setItemCarried” of the actor and set it to Null. Adds the entity back to the map in the location the actor is currently in. It finally calls the method addAfffordance for the target entity. The class also Has a description method that prints a description of the action.

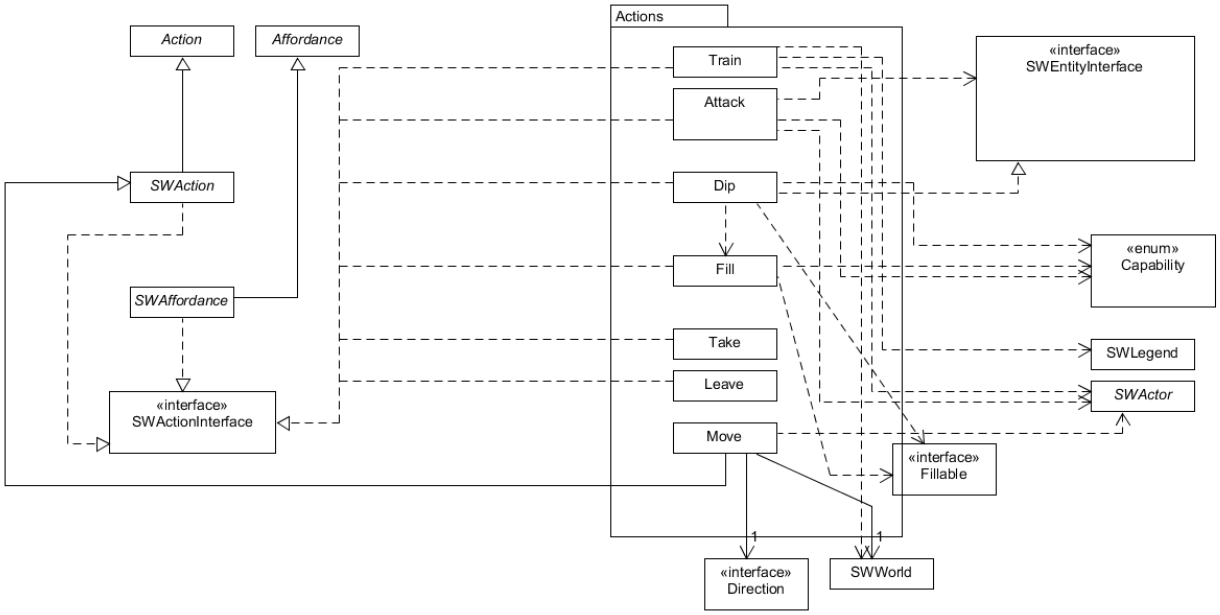
2. Force:



- We added an integer attribute called force in the class SWActor. This attribute keeps track of the actor's force level. If the force attribute is above a certain threshold, then the actor can use the force. We also added a setter and getter methods for retrieving and changing the value of an actor's force.

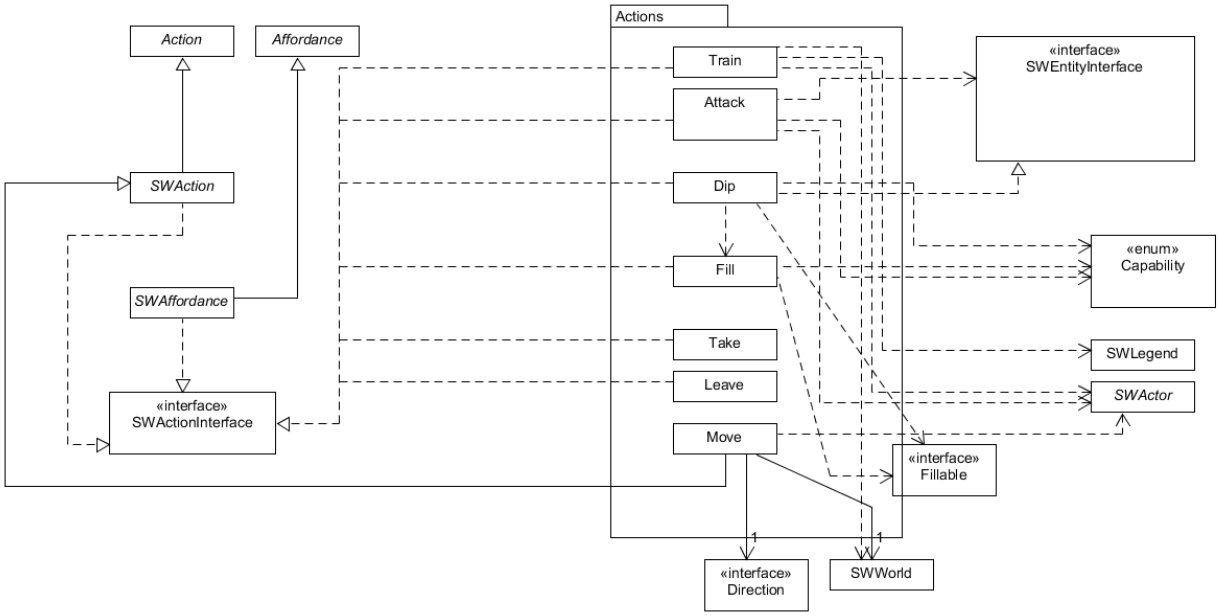
3. Lightsabres:





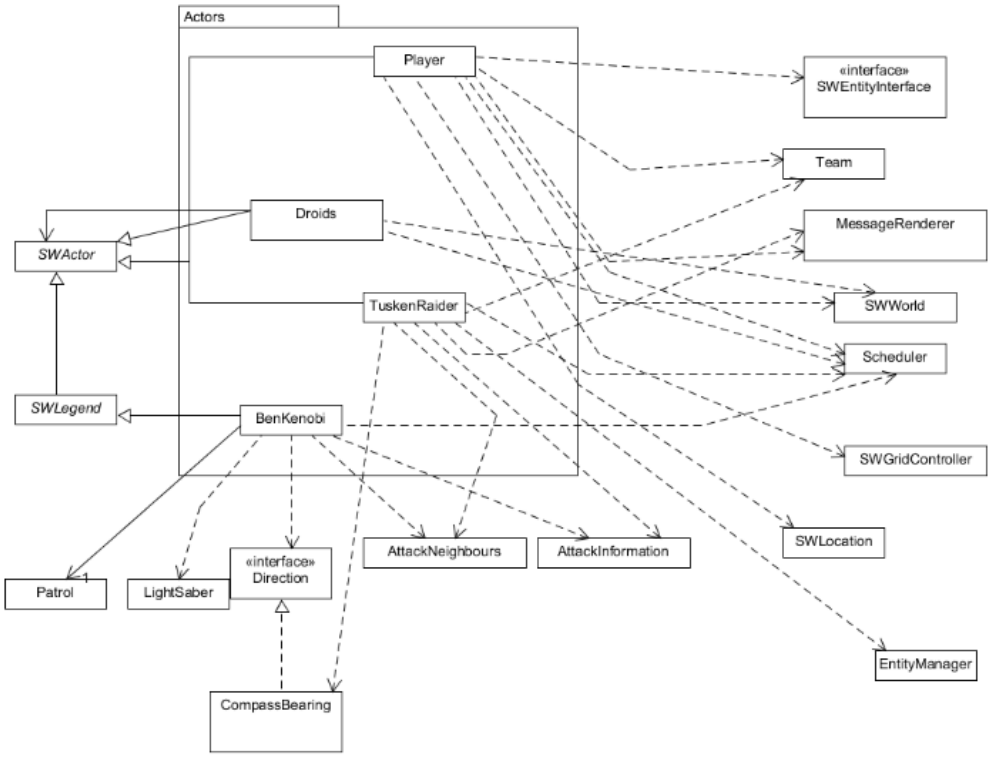
- We modified the Attack class such that if the actor is using a lightsaber and their force level is too low, 0 damage is inflicted. Else, the code executed.

4. Ben Kenobi:

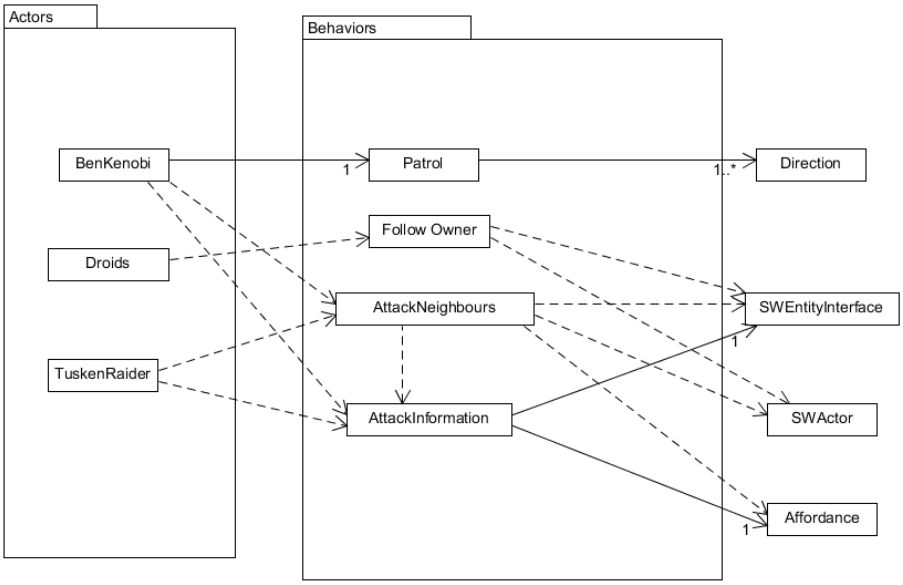


- We created a new “action” class called Train. This class checks the locations of Luke and Ben using the method of Entitymanager in the SWWorld class (`SWWorld.getEntitymanager().whereIs()`). If both of them are in the same location, then it increases Luke’s force level so that he’s able to use the force and lightsabres.

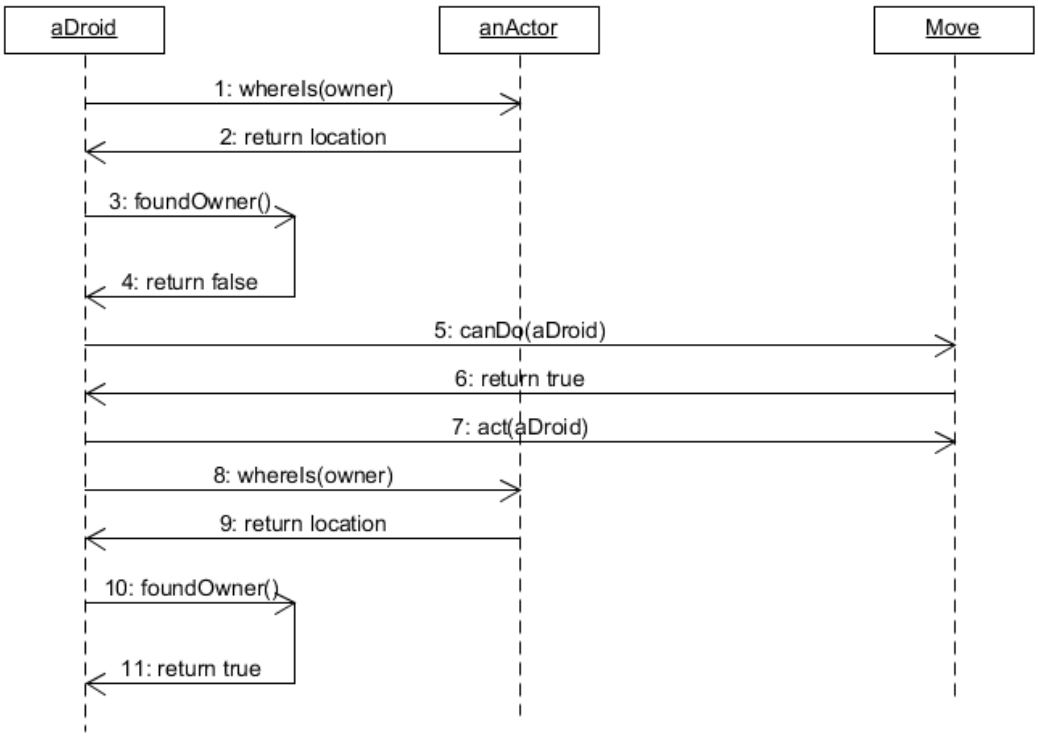
5. Droids:



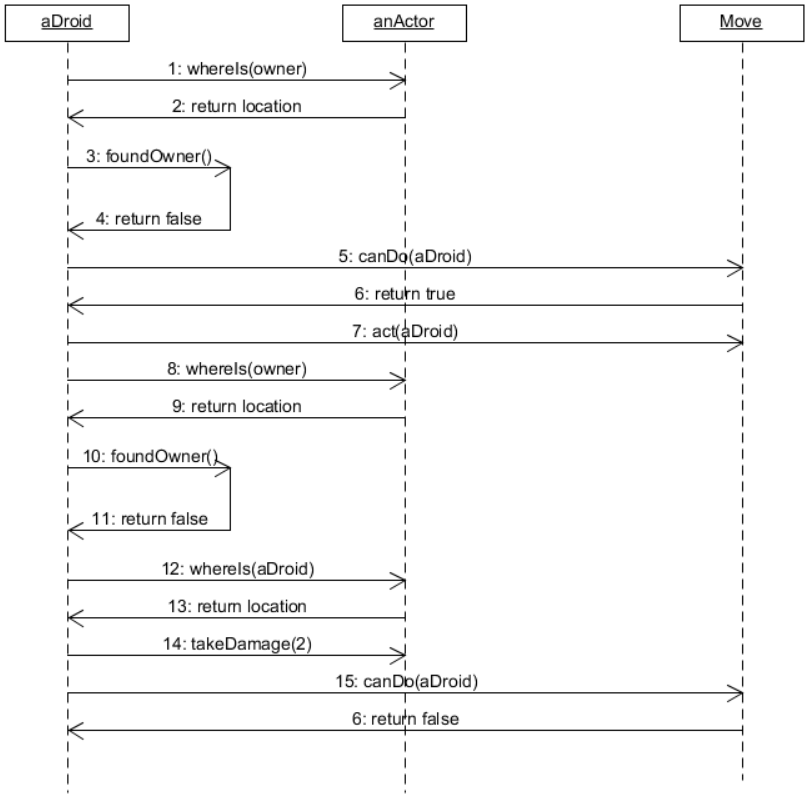
- Droids have been added as a Class in the Actors package and they inherit from the SWAcctor Class. We chose to do this because of the behaviours of droids. They have health like actors, they are able to move on their own like actors can and they can potentially attack like actors can. It made sense to place them with the actors for these reasons. There is an association between the Droids class and the SWActor Class because a droid belongs to a SWActor, and the droid must know who its owner is so it can follow them or find them on the map. There is a dependency between the Droids and the Schduler because the Droids movements must be scheduled using it when it is following or looking for its owner.



- Since droids are now actors, we must add a Class in the Behaviours package. This class is what will let us make the droids follow its owner or try to find the owner if it is lost for whatever reason. Droids have a dependency on this class because it's what enables them to be with their owners. FollowOwner is dependent on the SWActor class because it needs to know what actor the droid belongs to.



- This is an example of a sequence diagram for the process of a droid looking for its owner. We need to first check if the droid is with its owner, in this case it's not at first. We then have to check if it can move (if it has health points still or if it has reached a boundary), if it can then we make it move in a random direction. Again we look for its owner and see if they are in the same location, in this case they are, so the sequence ends.



- This sequence diagram is slightly modified, now we are checking to see if the droid is in badlands, in this case, we find that the droid is in the badlands and must take damage for it (in this case the cost is 2 health points). We continue to check if the droid can move and we find that it can't, its health points are too low making it immobile