# Force

We have implemented *Force* as an integer value in *SWActor*. This is because there may be other characters that can use the *Force*, not limited to just *Player* alone. One such example is *BenKenobi*. The initial force value of Luke is set to 25, and the initial force value of BenKenobi is set to 100(max value). There is also a MAX_FORCE assigned to 100 to represent the max value of a SWActor can possibly reach.

# Leave

1. The **getActorLocation()** method in original design is deleted

Instead of creating a separate method to get the actor's location, we decided to implement equivalent code inside the **act()** method, which invokes the **whereis()** method to determine where the entity target should be placed.

2. The implementation of *SWActionInterface* in the diagram should not appear in the diagram

*Leave* affordance extends the *SWAffordance* class, which implements *SWActionInterface.*

3. Associativity with *EntityManager* added

In the **act()** method, *EntityManager* is used to locate the carrier of the item.

4. Dependency with *SWEntityInterface*, *MessageRenderer*, *EntityManager* and *SWWorld* added

Instances of *MessageRenderer* and *SWEntityInterface* are passed into the constructor as a parameter, so there should be a dependency. When *Leave* class gets the location of a *SWActor*, the static attribute *EntityManager* is returned by the function **getEntityManager()** inside *SWWorld*, and the method **whereis()** in *EntityManager* is invoked.

# Train

We have decided to implement *Train* as a *SWAction*. By default, the *Luke* instance created in *SWWorld* will be initialized with *Train* added as an *Affordance* (this is because *SWAction* and *SWAffordance* both implements *SWActionInterface*, so they have a relation). However, the *Train* option will only be available in the text interface menu if *BenKenobi* happens to be in the same location as *Player*. *Train* will also decide if, after undergoing training, *Player* has sufficient

*Force* ability to wield a *LightSaber*. The constant FORCE_INCREASE_PER_TRAIN represent the increasing value of force in each training to avoid magic number.

# LightSaber

We removed the default **WEAPON** capability of a *LightSaber* that is initialized by its constructor. In its place, we have added a method, **canUseAsWeapon()**, that checks if the *SWActor* holding the *LightSaber* has sufficient force to wield it as a weapon. If they do, add a **WEAPON** capability to the *LightSaber*. This method will also be called in the constructor, as *BenKenobi* also possesses a *LightSaber*. The constant FORCE_LIMIT represent the lease force that can use lightsabre as weapon.

# Droid

1. *Droid* should extends *SWActor,* and *Own* should extends *SWAffordance*

We decided to use *SWActor* rather than *SWEntity* as the superclass of the *Droid,* as it has a hit points attribute like other *SWActors*, and it needs to move according to the *Scheduler*. Own should extend *SWAffordance* because it offers a more specialized interface for the game.

3. There are dependencies between *Droid* and *SWWorld*, *Scheduler*, *MessageRenderer* *CompassBearing* and *Grid*

The *SWWorld* and *MessageRenderer* are passed as parameters, and *Scheduler* is used to schedule a move event when a *Droid* moves. The *CompassBearing* is used to iterate through all possible directions. The *Grid* instance is returned by the method called in *SWWorld*, and the method **getLocationByCoordinates()** is called to determine the location of *BadLands*.

4. Removed redundant methods

The inherited methods **canDo()**, **act()**, and **getDescription()** are enough for the *Droid* class.

5. There are associativities between *Droid* and *SWActor* and *Direction*

*Droid* needs to take note of its owner to determine its behaviour. *Direction* should also be known, as *Droid* must know if there is an available direction while wandering around the map.

6. There are dependencies between *Own* and *SWEntityInterface*, *Droid*, *SWActor* and *MessageRenderer*

*SWEntityInterface* and *MessageRenderer* are passed as parameters in the constructor. *Droid* and *SWActor* are used inside **act()**.