**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*.

**Leave**

This is the documentation of several changes we made in leave class

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

Instead of create a seperate method to get the actor's location i decided

to put the equivalent code inside the act() method, which invoke the

whereis() method to determine where the entity target should be

placed

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

diagram

Leave affordance has extended the SWAffordance class, which has implemented

SWActionInterface

3. Associativity with EntityManager added

In the act() method, entityManager was used to locate the carrier of the item

4. Dependency with SWEntityInterface, MessageRenderer, EntityManager

and SWWorld added

Instance of MessageRenderer and SWEntityInterface are passed into

constructor as parameter, so there should be dependency exist

When leave class get the location of a actor, the static attribute

entityManager is returned by the function getEntityManger() inside SWWorld,

and the method whereis() in EntityManager is invoked. They all implies

dependency

**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*.

**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*.

**Droid**

1. Droid should extends SWActor not instead of SWEntity

We decide to use SWActor rather than SWEntity as the superclass of the

Droid. Since for Droid, it has hitpoints attribute as other actor, and

it need to take some movement while the time passing

2. Own should extends SWAffordance instead of Affordance

The SWAffordance offer more specialized interface for the game

3. There are dependency between Droid with SWWorld, Scheduler, MessageRenderer

CompassBearing and Grid

The SWWorld and MessageRenderer are passed as parameter, and Scheduler is

used to schedule the move event when Droid move. The CompassBearing is used

to be iterated through all the possible direction. The Grid instance is

returned by the method call in SWWorld, and the method

getLocationByCoordinates() is called to determine the location of badlands. They all implies dependency

4. Method in own class changed

The inherited methods canDO(), act(), getDescription() are enough for own

class

5. There are associativity between Droid with SWActor and Direction

Droid need to note the owner of itself along with it to determine the

behavior of itself. And Direction should also be remembered since it need

to stick with it whenever there is an exit in that direction while wondering

6. There are also dependency between Own class with SWEntityInterface, Droid,

SWActor and MessageRenderer

SWEntityInterface and MessageRenderer are passed as parameter in the

constructor. Droid and SWActor are used inside act().