Reservoir:

* Make use of hitpoints from SWEntity, set hitpoints to 40 when constructing reservoir
* Override the takeDamage method from SWEntity, make it so that when taking damage and hp is below 20, change the descriptions and symbol.
* When hp is 0 or below, change the descriptions and symbol to meet the specifications. Also, remove Dip affordance since a wreckage doesn’t have any more water.
* To deal damage to Reservoir, call the takeDamage method
* Since it cannot be attacked, the only way Reservoir can take damage is by Grenade, which will be implemented later
* Advantage: Easy to implement, only use methods and attributes from superclass
* Disadvantage: No additional features for other classes

Grenade:

* Create new entity Grenade
* Create new “Throwable” capability
* Create new Throw affordance and put it in Grenade
* Throw affordance will be an affordance that specifies if an entity can be thrown into a location and deal area damage
* The location for the throw action is the location of the actor, and then it will evaluate the locations around it and deal damage accordingly
* When initializing SWWorld, initialize multiple grenades at different locations
* Grenade will have a new attribute:
* int radius: the spread of the area damage
* Grenade will set its hitpoints to 20
* Grenade will have the Throwable capability
* This new capability will be checked by the canDo method of Throw, and canDo will return true only if the itemCarried of actor has the Throwable capability
* When act method of Throw is called, it will get hitpoints and radius of grenade, then gets the location of the actor calling the Throw.
* It will then start checking the locations around actor location in order of the radius
* So first it will start with radius 0 which means the location itself
* Then radius 1 will be the neighbor locations of actor location
* Then radius 2 will be the neighbor locations of those locations checked by radius 1
* And so on until it reaches the radius of the item
* For each of the locations checked, it will get all entities on the location
* Damage calculation will be item hitpoints/2r where r is the radius
* After damage calculation, check if the entity’s hp is 0 or less
* If it is, remove Attack affordance from that entity (if they have it)
* After every locations are checked, remove the item carried by actor for good by setting it to null
* The grenade example, it will check the actor location and will damage every entity except for actor in that location by 20/20 = 20
* Then it checks locations that can be reached in one step from the location, and will damage every entity except for actor in that location by 20/21 = 10
* And then it checks locations that can be reached in two steps from the location, and will damage every entity except for actor in that location by 20/22 = 5
* If the entity has 0 or less hitpoints after the damage calculation, remove attack affordance from it
* After that, setItemCarried by actor to null
* Advantage: Making throw to consider radius will provide extensibility for future developments of items with greater area explosion. Also, it includes the fact that dead actors cannot be attacked.
* Disadvantage: The name of the affordance “Throw” can be misleading. Its better to name this affordance “Explosion” because of the nature of dealing area damage, however we’re sticking to this name because making a new Explosion affordance and connecting it to Throw would be redundant.
* Future development: If there’s more entities that can be thrown, consider creating a new abstract class “Projectile” that will have the Throwable capability and radius attribute instead of adding those one by one to each Throwable entity.

**Jawa SandCrawler:**

**Enter and Exit Affordances**

* Added two new affordances Enter and Exit that take SWEntity target to teleport to
* Using Enter Affordance will cause the Entity that used it will be teleported to SandCrawlerGrid’s door in the SandCrawlerGrid.
* Using Exit Affordance will cause the Entity that used it will be teleported to SandCrawlerGrid’s position on the main map.
* Check SWActor for its force to be able to use Enter or Exit affordance.
* Check whether the one that use act is Player or not

If it is player change the grid to a grid where the player is.

**SWWorld changes**

* Make a new Grid attribute in the SWWorld class called SandCrawlerGrid and currentGrid.
* currentGrid’s default value is myGrid (main map).
* currentGrid will be used to store which Grid the actor is on.
* Modify the *getGrid()* method to return currentGrid instead of myGrid.
* Create a new Method called *getSandcrawlerGrid()* to return SandCrawlerGrid.
* SandCrawlerGrid will be constructed as a new map in SWWorld.
* Make a new Constructor in the SWGrid that can take an x,y value instead of 10,10
* Add a new SWEntity Door and attach Exit affordance to it and place it inside SandCrawlerGrid for player to exit.
* Added a loop to hard code all SandCrawlerGrid location’s *shortDescription()* and *longDescription()*

**SandCrawler Class**

* Make a new class called SandCrawler that extends from SWActor;
* Added an atrubute Patrol class to the SandCrawler class from BenKenobi.

SandCrawler only move on the second turn, so we add counter for the solution

* In the method *act()* we check for every entities on the spot

If there is Droid Entity, it will move the droid to SandCrawlerGrid

**SWGridTextInterface**

* Added a *setGrid()* method to change what grid is going to be displayed

**SWGridController**

* Added a SWWorld world attribute to call a method from SWWorld
* In the method *render()* it will always call *getGrid()* from world to check current active grid and then set SWGridTextInterface’s Grid to the currentGrid

Advantage : we can add many more new grid as it is flexible to change to new Grid (map).

Disadvantage : Enter and Exit is hard coded to automatically enter SandCrawlerGrid and exit to mainGrid. And it use SWActor to change the world’s grid

Future Development : as you can add more grid (map) we can use list to change the grid accordingly, rather than hard code it only to move to SandCrawler Grid.