REQ1: Travelling between Maps

The diagram represents an object-oriented design for requirement 1 of assignment 3 and displays the necessary classes and their relationships with each other in order to fulfil the needs of the overall system. In order to fulfil the features outlined in the requirement, a total of three new classes have been added to the system. These new classes make use of pre existing code from the engine and game, allowing us to easily extend our system.

The first class that has been added is the Cliff class. This class extends Ground and will kill the player on the turn he or she steps on it. This is achieved by overriding two methods in the class that are inherited from the parent Ground class. The first overriding method is the canActorEnter method; here we have added code to perform a check to see if the actor is the player before allowing it to enter, preventing enemies and npcs from walking into cliffs and killing themselves. The other overridden method is the tick method; here we have added code to create and execute a new DeathAction on the actor standing on the cliff, allowing us to effectively kill the actor at the end of the turn. Overall, this approach applies the DRY principle, as by extending the Ground class, we avoid repeating code, as well as the open closed principle, as we have extended our Ground using the new Cliff class to implement this feature instead of modfying it. Additionally, this fulfils the Liskov substitution principle as well since Cliff objects can be used in place of parent Ground objects. An alternative to this implementation would have been to override the tick method to deal damage or attack the player for a large amount of damage; however, this is not an ideal implementation as attackAction would need to be modified to be able to accept a Ground as an attacker, which violates the open close principle.

The second and third classes that have been added are the GoldenFogDoor class, which extends Ground, and the TravelAction class, which extends Action, respectively. In order to allow the player to travel to other maps using GoldenFogDoors we have overridden its allowableActions method to pass a new TravelAction to an actor that has the CAN_TRAVEL capability. This TravelAction, when executed, will move the player to a specific location on the map that has been encoded in the GoldenFogDoor. This implementation fulfils the DRY, open-closed, and LSP principles as both classes utilise and extend existing classes from the engine while still being able to be used in place of their parent classes. Furthermore, this implementation also fulfils the single responsibility principle as we have split the responsibility of transporting the player into two separate classes, one to provide the action and another to facilitate it, ultimately resulting in less coupling. The drawback of this design is that now we must manage two separate classes instead of one and store the location we wish to travel to in each object instance, ultimately resulting in more verbose code. The alternative would be to modify the GoldenFogDoor's tick method to simply transport the player to its designated location at the end of the turn. This implementation would reduce the number of classes to just the GoldenFogDoor, however, this breaks the single responsibility principle as well as couples travelling very tightly to the GoldenFogDoor class, making it difficult to extend in the future if there are more cases where the player can travel around different maps.