Design Rationale for Requirement 5

Player Death

In order to trigger the changes in the entire map when the player dies, a respawn method has been created. This method is responsible for changing the attributes of the player and calling the MortalRespawn interface (explained below) to trigger the changes in the rest of the players. The runes of the player are also removed from the inventory and dropped at the death location. It also sets the location of the player to its beginning location.

The respawn method is called in the player's unconscious method, overridden from the parent class.

This implementation is not extensible as if other actors could trigger the respawn, the methods could not be reutilised and would violate DRY too. However, this implementation works as there is only one main actor so it was completed under the assumption that the player would be the primary entity to trigger the respawn.

Mortal Respawn Interface

To allow necessary entities to be notified and undergo the required changes, the observer principle pattern has been utilised. The parent Respawn class stores the ArrayList of all the observers and is implemented by the Mortal Respawn class. The MortalRespawn class is responsible for registering, unregistering and notifying the entities.

A Respawn Entity interface has been created which is implemented by all entities that are affected by the player's death. All these entities have a respawn method that is called through the notifyEntity method and carries out their specific responsibilities.

For the enemies, it is added in their constructor so that once spawned, it automatically becomes a part of the list. For items such as runes, since they are mobile and portable, the tick method is utilised to check whether they are in an actor's inventory or on the ground and are removed or added respectively.

The first implementation of the observer principle led to the creation of another subclass of Respawn, ImmortalRespawn for entities such as gate which could only be registered once. This was due to the fact that an error occurred every time an enemy or item was unregistered within the respawn method. The only other way to remove an enemy was by clearing the list, hence the gate implemented ImmortalEnemy instead and the list was not cleared for its type. However, due to the increase in complexity and lack of efficiency with this approach, it was scrapped.

For the second implementation, we figured out that the unconscious method of the enemy can be used to unregister the subject. This meant the entities could be unregistered without the need for the ImmortalRespawn, hence being a better approach.

For the respawn method in Enemy, since the enemy's map and location is not passed through the method, the hurt method is used to inflict damage equal to the actor's current HP. This implementation has one downside which cannot be fixed due to the limitations of the engine, if the player is killed by the enemies, some enemies who have already finished their turn only have their unconscious method checked in the next turn. This means they require an extra turn to get removed from the map.

This approach follows the SRP as each entity is only responsible for itself in the case of a player respawn. This approach also satisfies the Open/Closed Principle as no previous code was modified and remains open for extension in case new entities need to be added.
The implementation although currently follows DRY as all changes for the enemies are grouped in the enemy item class, it will not adhere to it if the implementation extends to items and weapons.

Unconscious method when the player is attacked by another actor is shown in the Sequence Diagram for the Requirement 5. It demonstrated the whole respawning process for the player as well as notifying all entities to refresh the map.