REQ 2 Design Rationale

The diagram represents an object-oriented system for the runes system and trading within a rogue-like game. This includes multiple concrete classes working together.

The concrete class RunesManager is introduced as a way to manage all runes functionalities and interactions within the application. This includes updating the number of runes a Player may have after killing an enemy. The Runes interface is also implemented to help out with tracking the number of runes of each Actor. Its sole purpose is to update and return the number of runes a player may have (Single Responsibility Principle).

The Runes interface is also implemented to avoid having to change implementation of the Player and Enemy classes if RunesManager were to receive a change or update to the class which may include the methods on calculating how runes are calculated (Dependency Inversion Principle).

To implement the Trader, we had to consider which items were buyable and sellable - the GrossMesser can only be sold to the Trader but cannot be bought from him. Therefore, to distinguish between which items can be bought and sold we created two interfaces for buyable and sellable items - as opposed to a singular tradable interface (interface segregation principle). However, having two interface classes for buyable and tradeable can be considered an overapplication of SRP, as the classes may get too small. In our implementation, in terms of classes, they will be from the players POV, as the player can sell the Grossmesser but cannot buy it.