Assignment 1 Design Rationale: CL_AppliedSession10_Group1

Requirement 3 – Pot of gold, jar of pickles, drinking puddle

This design rationale considers implementation of the pot of gold item, jar of pickles item, and extension of the puddle. All new extensions of this requirement are considered as a form of consumable, as the design purposes require these items to be 'consumed' by the actor. Consumption in this requirement doesn't (necessarily) imply physical consumption; rather some individual logic that occurs when using the object. We implement the consume method in various ways between these new extensions: once by calling the add actor balance method for pot of gold, another by adding health for pickles or even 'modifyAttributeMaximum' when drinking the puddle. Refactorisation of consumeby method wasn't necessary as original code base from team mate already considered alternative consumption design.

DRY and singular responsibility principle are unanimously confirmed in this requirement approach as we extend old code through inheritance. This is evidenced in utilisation of the consumable and consume action of our previous codebase. As such we don't need to create duplicate classes but instead reuse code from the existing infrastructure.

Another principle evidenced in this requirement is readability. Code readability ensures that the quality of code is easily understood, modified, and maintained by other developers. This is evidenced through or code base localising item vales in a specific location within the class (as attributes), as opposed to spreading it all in different methods and injection.

Our approach is advantageous since it utilised previously made code from the consumable class, which made it relatively easy to extend without modification. This highlights the presence of the open closed principle, and how it can make an actualised different in the approach later in time. If the original code base generated consumeby method of fruit logic in the consumable method, this would have caused us to refactor old code. This implies poor code robustness and reusability. Further, all behaviours that align with the requirements are inherited. New similar items can follow the same procedure, making extending more items easy.

An alternative approach may consider better naming conventions. As it currently stands, our current implementation uses a randomised float value to get the chance that the pickles are expired. On one hand it may be argued that our current design considers connassence of meaning by reducing the amount of 'magic numbers' that exist representing variable value.

However, an alternative design may have more clarity in the naming conventions such that this is more comprehensive. On aspect to consider is through random logic which is shared ubiquitously amongst various classes. This may better implement DRY. Moreover, since values like expire chance and date that belong to a JarOfPickles they are initialised within the class instead of being injected through constructor injection. This causes the values to not be modifiable without adding code in the future, which violates the OCP. An alternative approach may abstract this sort further consumable logic into an abstract class or through the use of an interface.