# Engine Recommendations

Problem 1

Item class is dependent on Action class. For any new items it had to extend the item class

**Explanation:**

When creating new items, they all had to extend the Item class. For example sutring this assignment many items had different functionalities which were not supported by the engine such as: having nutrition for Food objects, being able to upgrade a ZombieLeg/ZombieArm object to a ZombieMace/ZombieClub object and keeping track of how much Ammunition is in a gun or ammunition on the ground. To implement this functionality, we needed to add abstract methods into the ItemInterface which lead to down casting these methods into other items that did not need them.

**Design Change:**

Have the class also iterate through items on every tick so there is less reliant on the Actions class and the expand the Item class to support this change.

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| * Reduce dependencies as much as possible principle is maintained. * Interface segregation principle is easier to maintain as unneeded methods do not need to be added to the item interface to allow subclasses to work | * This change may be hard for new users to recognise when to create a new item or a new action * Increases runtime of the software as now the items and actions need to be checked for what they can do. |

## Positive opinion 1:

Classes in the engine can act as superclasses which can be replaced by subclasses without breaking the application.

**Explanation:**

Classes in the engine all have methods that include parameters required for functionality for later implementation. Thus, the Liskov Substitution Principle is demonstrated withing these classes. The methods within the classes of the engine are very helpful for further implementation of features such as the tick() method in Items which allows that class to virtually keep track of time passing in the system. Furthermore, as these subclasses can be used in place of its parent classes it is much easier to encapsulate certain functionality to only work with specific types of classes without losing the functionality that the parent classes provide.