FIT2099 Assignment 3 (Recommended Changes)

Problem 1

What problem you perceive:

In the assignment, we were tasked to add endings to the game. The best way that we came up with was to add this functionality into World, however we are not allowed to change World class directly, as we are not allowed to make any changes to the codes within the Engine package. This means we have to subclass World in order to add additional functionality. By subclassing World class, we have essentially made World class itself obsolete. We have to also break the Don't Repeat Yourself(DRY) principle just to implement the required functionalities.

The design change you propose to address the problem:

Move World class to the game package.

The advantages, and any disadvantages, of the proposed change:

By moving World class to the game package, we can make changes to World directly. This also means that we do not need to subclass the entire World class just to make small changes to the code. This in turn makes sure that World is the only class that controls the game logic, rather than some subclass of it. This would also mean that implementing new features would not require us to break the DRY principle.

Additional Notes:

To add endings to the game, we can alternatively implement it within AttackAction. However, this implementation would break the Single Responsibility Principle (SRP), as ending game functionality should be handled by World class which runs the game.

Problem 2

What problem you perceive:

Action class does not have a way to calculate probability. This means that if we want to implement a functionality that involves any sort of probability we will need to instantiate a new Random class whenever we want to use probability. This means that we have similar code in multiple subclasses of Action class. This breaks the Don't Repeat yourself principle, as we are repeating ourselves due to this lack of functionality.

The design change you propose to address the problem:
Create a Random class object and allow access to said Random object.

The advantages, and any disadvantages, of the proposed change:
By creating a shared Random class object, we can just use the shared Random class object to compute any probability calculations required. This also helps us to fulfil the Don’t Repeat Yourself principle, as we do not instantiate a new Random class object in subclasses that need to compute probability calculations. However, if no probability calculation is required in all subclasses of Action, the creation of said Random class object would become useless and take up space.

<u>Problem 3</u>
What problem you perceive:
Weapon and WeaponItem classes do not have an accuracy variable. This means that in order to allow for weapons to miss, you have to create a variable in every WeaponItem and Weapon class that stores the accuracy of the item. This means that there are multiple subclasses that have accuracy variables. This means that we are repeating ourselves, which violates the Don’t Repeat Yourself principle. Classes such as IntrinsicWeapon do not allow us to modify their code per assignment rules, so their accuracy is stored within Zombie class. This means that we violate the Single responsibility Principle to implement an accuracy, as the accuracy variable is not stored in IntrinsicWeapon class.

The design change you propose to address the problem:
Weapon or WeaponItem classes have an accuracy variable ranging from 0 to 100 that is required in the constructor, along with a constructor that does not require an accuracy value to be inputted. This means that any weapon without an accuracy value given has an accuracy of 100.

The advantages, and any disadvantages, of the proposed change:
By having an accuracy variable, it allows players to implement hit chances on the weapon. This means that we do not need to violate the Don’t Repeat Yourself principle in order to implement accuracy. By having another constructor that does not require an accuracy value, we can also ensure that any implementations that do not require accuracy can be implemented easily. However, any implementations that do not require accuracy would be having a redundant variable attached.

What positives you perceive:

<u>Positive 1</u>

The creation of the Capability class made our coding experience better, as we can assign capabilities to all Actor and Item objects. This made classifying and coding specific functions for specific Actors a lot easier and cleaner, as we can just check it's available capabilities to identify the object instead of checking if it's an instance of something. This also allows us to create actions that are only allowed based on certain capabilities available (eg: you can only eat food objects. To check this, we check if the item has capability food).

<u>Positive 2</u>

Action class(systems) are well implemented. Action class is implemented as a base class for all other Actions which greatly reduces the amount of duplicate codes and improves the maintainability and readability of the code. Also, a single responsibility principle is encouraged as each individual action can be implemented as a single class easily by inheriting the Action base class. By implementing as such, it allows the programmer to easily fulfil the DRY principle and Single Responsibility principle.

<u>Positive 3</u>

Exit class is a very practical and useful class. Exits allow programmers to easily scan for valid/ available spots to take an action. It also allows you to check for direction, which helps a lot when implementing functionalities such as running away and hunting. It also reduces the amount of work significantly, when implementing the Shotgun (scan for valid direction to shoot).