Recommendation for changes to game engine

Action class

- Action class in engine package is an abstract class with execute, menuDescription and hotKey methods
- Most of the classes extending Action class only return empty string (“”)., it is redundant to override the method everytime we extend Action class
- The hotKey method in abstract class Action can declared to return empty string by default, so that we do not need to override hotKey method in every subclass
- This can reduce redundancy of code and follow design principle “Don’t Repeat Yourself”
- This method can be override if it needs to be declared as string other than empty string

AttackAction class

- In AttackAction, the probability of hitting target Actor is set in execute method, where the Random value is declared locally in execute method. AttackAction in engine code allows us to change the verb for attacking action, so it will seems like different attacking action.
- However, since the probability of hitting is declared locally in execute method, if we want to have a different attacking action with different probability of hitting, we will need to generate a new Action subclass and so similar action asAttackAction in order to change the probability of hitting
- For instance, in our game, we need to have Water Pistol to have 70% chance of hitting YugoMaxx, and we could actually use AttackAction to change the chance of hitting. However, since the probability is declared in execute method, we need extend Action class to override it to generate the probability and rewrite a similar code as AttackAction, which is quite redundant.
- The random value in AttackAction could be set as class level attribute, and we can have a setter to set its value from other class. If this were done, we could use AttackAction in our YugoMaxx without ShootingAction class which does similar things and redundantly coding out the same execute method to change the probability of hitting
- However, the disadvantage of setting random value as class level attribute will violate design principle of declare things in the tightest possible scope and will increase the dependency.

Menu hotKey system

- The key used in the menu does not have central management system, so when player has more than 20 items in their inventory, the menu keys will be override (6 keys , causing some of the actions cannot be displayed in the menu since there is only 26 alphabets key can be displayed along with the actions to be performed.
- Moreover, it is confusing for player to use different menu key while playing. If there are too much items in player’s inventory, player needs to look through each abd every option to search for the action they desire to perform.

- It is better to have a fixed keys for each action. For example, we can have key for player to open their inventory, and player can select items to be dropped after the items in their inventory is displayed in a new menu which only shows the item to be dropped , so more options can be displayed. We can have a fix ‘q’ key for quit game which makes more sense for player.
- This kind of menu is also more user friendly and can prevent player from accidentally pressing quit game option if the keys for actions are fixed
- However, if there are too many different actions to be performed, the key will still be override, but it is better than displaying all variables of same action (e.g. show all items in inventory to be dropped in main menu) which waste more keys in menu.

Item Class

- The PickUpItem action is added to the allowableAction of the constructor of Item class. Hence, if extending Item class, all item will be forced to have PickUpItemActions which doesn’t make sense if the item is unpickable.
- In our implementation, we built a rocketPad and rocket by extending the Item class, hence we need to remove the PickUpActions again by overriding the getAllowableActions to clear the actions list again.
- This is redundant as we added it and we remove it again hence it is better that if we initialise allowabaleActions to a list of empty actions.
- The better approach is to remove the “static” and new item creation from the newFurniture class and newInventoryItem class. This will allow the new Item type of either newFurniture and newInventory to be created in the constructor of Item class.
- So, we can create all the different types of item by passing only name and displayChar into the constructor of Item. After create the item, we can invoke the method of newFurniture or newInventoryItem to set the allowableAction of the item according to the item’s type.