FIT2099 Assignment 3

Recommendations & Feedback

Arie Hendrikse & Rahn Stavar

As a whole, we enjoyed working with the FIT2099 game engine and the existing Zombie World game code to implement additional features.

Through our experience gained working with this engine and the existing game code over the course of our three assignments, we would like to propose some suggestions for future versions of the engine and game code, as well as discuss some things we particularly enjoyed about it.

# Positive Feedback

The Actor, Item and Ground classes made adding new things to the game super easy, letting us simply extend the classes we needed. This helped greatly to reduce repeated code and ensured everything we added worked well within the existing system. The tick method also made it easy for us to not have to work out our own logic for this.

The Display class worked well for printing to the user and we enjoyed the style of the game. The fact that the object oriented design of all the engine classes meant we did not have to worry about working out dependencies and the design diagram was a major help.

We found the fact that complex classes such as item, and Action already being implemented gave us a clear set of design principles to follow for us to work with. This meant that we wouldn’t get to far into our work before realising that we didn’t implement something major that was needed and avoided us having to start from scratch when already through the project (this did happen, just not on a major whole game scale).

We appreciate the work that the FIT2099 team put into this engine, it has given both of us a great understanding of OOP that we will find valuable for the rest of our programming career. We also found the game super fun to play and it enticed us into challenging ourselves to add bonus features or special Easter eggs (old mcdonald) just because we could.

# Our recommendations

## Items Cannot Print Messages when they Tick

Perceived Issue:

Items may need to be able to print messages to the display when they tick. For example, a corpse may want to print “Jack rose for the dead as a zombie” or items that hurt actors (such as the turret in our bonus marks features) may want to print “Turret shot Jack”.

Currently there are two options for doing this, and neither are perfect. Option 1 is to just print the message using System.out.println(), however this isn’t good design as the game uses a display class for printing, which would be upgraded in the future to not print messages to the console, but instead to a GUI. Option 2 is to pass the item an instance of the display class when it is created, which it would then store and use when it needs to print, but this creates an association with the display class so this isn’t great either.

Recommended Fix:

A possible fix for this issue is to pass the display into the Item class’s tick method. This could be done by modifying the Item engine class, and then changing the World, GameMap and Location classes to pass this display parameter into its tick calls.

This would allow items to then print messages to the display when they tick, solving the issue outlined above.

A perceived downside to this fix however is that 4 engine classes will need to be modified, and it would change the tick method signature so existing games that override the tick method in any of their items

would need to be modified to make their overridden tick method signature match the base class’s tick method signature.

## Naming of the Generic Actor class

### Perceived Issue:

ZombieActor is a confusing name – at first we didn’t realise it was a generic actor class for the zombie game, and instead thought it was a class for representing a zombie (which was confusing as there was also the Zombie class).

We understand this issue is specific to the zombie game but we can also imagine this issue may happen in future game adaptations, such as in a Harry Potter game there may be a HarryPotterActor class (the generic actor class) and a HarryPotter class (the class representing the character Harry Potter), which would present the same issue.

### Recommended Fix:

Rename the ZombieActor class to “ZombieGameActor”, or in future variations of the game ensure the name of a character in the game is not used in the name of the game’s generic actor class.

This would make the responsibilities of the game’s generic actor class clearer from the beginning, as new developers are getting used to the engine and existing game code.

## Allowable Actions can be Used while the Item is on the Ground

### Perceived Issue:

There is currently no easy way to stop an item’s allowable actions from showing to the player when the item is on the ground (not in the player’s inventory). This meant we had to use the tick methods to check if an item is on the ground or not, and then add or remove actions from the item’s allowable actions accordingly every turn, to stop the player from being able to craft limbs into better weapons while they’re still on the ground (the assignment specs specified that actors could only craft weapons if they were in the actor’s inventory).

This workaround worked well but we think the engine could be improved to accommodate this scenario natively.

### Recommended Fix:

A possible fix for this issue is to add a new Actions attribute to the engine’s Item abstract class, such as “protected Actions carryingActions” and add a getter for this attribute, such as getCarryingActions().

Then, change the World class’s proccessActorTurn() method to also include actions in this list in its returned actions list, but only when the Item is being carried. This could be done by adding a line to the first loop in the proccessActorTurn method, the loop that scans the actor’s inventory. The new line could be something like: actions.add(item.getCarryingAllowableActions).

This fix would allow developers to add actions that should only be available when the item is carried to an Item. In our case, we would have been able to add the CraftAction in our Limb class (which extends Item) to the carryingActions attribute instead of adding and removing the CraftAction from the allowableActions attribute each turn depending on whether or not the Limb was on the ground or in an inventory.

The only perceived disadvantage to this fix is it may make understanding how to add an action to an Item more difficult, as the developer will need to understand the difference between the Item class’s carryingActions and allowableActions attributes. We believe adding proper documentation to both attributes would mostly alleviate this disadvantage. allowableActions could also be renamed to make this difference clearer.

## Cannot Currently get All Actors on a specific GameMap

### Perceived Issue:

The GameMap class in the engine package currently allows you to find an actor on its map, and check if there is an actor at a specific location on the map, but there is no way to get a list of all the actor’s on the map or all the locations on the map with actors.

This meant when we needed to display a list of all the possible targets when using the sniper rifle, we had to loop through every location on the map and check if there was an actor there, which could get inefficient for large maps.

### Recommended Fix:

It would be useful if the GameMap class had a public method, such as getAllActorLocations(), which would return a copy of the GameMap’s actorLocations attribute.

This would have allowed us to use the ActorLocation class’s iterable functionality to loop through all the actors on the map, in the same way World’s run() loops over every actor every turn. This would have been much more efficient and would have made implementing the sniper rifle slightly easier.

A possible disadvantage of this fix is that it adds another public method to the GameMap class’s public interface and that new method will always have to return an ActorLocation instance so that no existing code that relies on this method is broken if the GameMap class’s private implementation changes in the future to not use an ActorLocation instance for keeping track of its actors.

This seems like an unlikely scenario as ActorLocation is used widely throughout engine and we see no reason for it to change, so we think it is a worthwhile trade-off for this new functionality.