FIT2099 Assignment 3 report

Problems

Cannot execute actions ‘silently’

This was a problem especially we implemented Fish as an Actor. Fish has a huge population and they wander around and every time they do so, a line is printed: “Fish wanders around”. This clutters up the display and is annoying because Fish is not a main actor in the game. The method execute() in Action cannot return null because it is called in processActorTurn() in World and is a null return is not handled there.

I think the easiest way is to check for null return in processActorTurn() and if null, do not print anything.

This improvement allows control over what action messages should be printed on the display, for example maybe we do not want to print anything when the dinosaurs are just wandering around.

Getting reference of the Actor player

The World class keeps a reference of which Actor it deems to be a player but we do not have access to it in the client code. It would be great if we can get it, especially for instances where we needed to do the check “instance of Player” to give actions that are specific to the player only. The instance of checks are fine but we are treating the Player class as something special and we are depending on a concrete class. We will need to modify these checks when we want to change to another class.

The places that we need the reference to player is generally in getAllowableActions() in Actor, Ground and Item. These methods are called in World so the simplest way is to modify these methods’ signatures and pass in the reference of player.

Item doesn’t not have reference to the Actor, Location or Map in getAllowableActions()

This was not a huge problem for us during development but it limits the flexibility of item to give Actions depending on the Actor or the Ground it is on. One way that this limitation was shown is that we cannot let items to offer the EatItemAction of itself because it cannot check for if the Actor can eat it or not.

I think one way to fix this is to split getAllowableActions() into 2 methods, one for Actor and one for the Ground/Location.

Actor carrying items also has the items' skills

This was a problem for us when we implemented FoodSkill to know which items can be eaten by a carnivore, herbivore or marine dinosaur. If an actor (e.g the player) is carrying a food item, they also become food and will be attacked by the hungry dinosaurs. For this implementation, it was cumbersome to not let the Protoceratops attacking the player when they carry the herbivore food item while allowing the carnivorous dinosaurs to attack the player.

The problem in general is that the current implementation of the engine does not differentiate between an Actor's intrinsic skills and skills from the items they are carrying. I think one way to tackle this is to modify the Skilled interface to have methods like hasIntrinsicSkill() and hasExtrinsicSkill().

Action menu

We wanted to implement submenu for actions like buying and selling at Store to avoid cluttering the main menu (the menu is also limited by 26 alphabet characters and 10 numerical characters). It is possible to create an Action to show the submenu but it would require the Action to instantiate an instance of Display, which means that this would have to be modified if we wanted to swap with another display class in the future.

One way to fix that is to pass the display as a parameter in allowableActions() method in Ground:

```
51
52      /**
53       * return list of BuyAction and SellAction for different items
54       */
55      @Override
56      public Actions allowableActions(Actor actor, Location location, String direction, Display display) {
57          Actions actions = new Actions();
58          if (actor instanceof Trader) {
59              createItemList().stream().filter(item -> item.isBuyable())
60                      .forEach(item -> actions.add(new BuyAction(item)));
61
62              actor.getInventory().stream().filter(item -> item.isSellable())
63                      .forEach(item -> actions.add(new SellAction(item)));
64
65              if (!actor.getTaggedActors().isEmpty()) {
66                  actions.add(new SellTaggedActorsAction(actor.getTaggedActors()));
67              }
68          }
69          Actions returnActions = new Actions();
70          returnActions.add(new MenuStore(menu, display, actions));
71          return returnActions;
72      }
```

Figure 1: allowableActions method in Store class; display is passed in as parameter

```java
package game.actor;

import java.util.List;

public class MenuStore extends Action {
    private Menu menu;
    private Actions actions;
    private Display display;

    public MenuStore(Menu menu, Display display, Actions actions) {
        this.menu = menu;
        this.actions = actions;
        this.display = display;
    }
    @Override
    public String execute(Actor actor, GameMap map) {
        Action action = menu.showMenu(actor, actions, display);
        return action.execute(actor, map);
    }

    @Override
    public String menuDescription(Actor actor) {
        return "Store menu";
    }

}

```

Figure 2: MenuStore Action used to show sub-menu

It would be even better if this functionality is supported within the Menu class for better encapsulation.

World and Display

World has an association with concrete Display class. I think it could be better to do a dependency inversion here and let World depends on a Display interface. This way we can swap out different implementations of the concrete display classes, say we want to display to different parts of the game to different windows (i.e one window for the map, one for the action menu,etc.). However, we would need to have a good design for the public methods of the Display interface that will not be needed to be changed in the future for an easy swap.

Good design

GroundFactory

This is an example of dependency inversion: GameMap depends on GroundFactory interface instead of the concrete FancyGroundFactory. This means that we can change implementation/swap out FancyGroundFactory if needed as long as it supports the newGround() method.

FancyGroundFactory

This allows easy expansion of more ground types (Open-closed principle). However it can be confusing as the new Ground must have an empty constructor for this to work, so better documentation would help.

Interfaces

Some interfaces in the engine are: Skilled, Printable, Weapons.

For Skilled and Printable, they are implemented by Actor, Ground and Item and this reduces code repetition. For example, the Display class is used for Actor, Ground and Item because the Display class depends on Printable. This means that there is no need for separate Display classes for each of Actor, Ground or Item. The Skilled interface is implemented by the Skills class, which is instantiated as an attribute in Actor, Ground and Item (which also implement Skilled). This reduces repetition and hides implementation of how the skills are stored and checked.