FIT2099 game engine changes

Having access to the allowable actions of the current ground

Currently, when an actor is standing in a particular location, they are able to access the allowable actions of all of the ground types of exits surrounding them. They are not able to access the allowable actions of the current location they are standing on which would be a useful feature for creating actions where an actor must be standing on top of a particular type of ground.

This limitation is in the processActorTurn method in the World class where the exits of the current location of the actor are iterated through to obtain the allowable actions, but there is no check for the current location's ground. The fix would simply consist of adding a few lines to add the allowable actions of the ground the actor is standing on into the available actions for the actor's turn.

This change would offer the advantage of being able to interact with ground of the actor's current location which would make sense, but also allow for some types of ground to have special interactions which require actors to be standing on top of the ground rather than next to it. There are many potential features which may meet this requirement, and making them simple to implement increases extensibility.

Difficulty of Testing Code

It's really slow to test a lot of code in the game. Breeding, as an example, was really challenging as there were so many conditions that needed to be met in order for an end-to-end test to complete. The flow of the game is incredibly slow and even moving from one side of the map to the other can take several minutes. Complex interactions between Actors in the system can take ages to appear, and as such edge cases that cause bugs can be hard to encounter.

This is partially due to the nature of the game, any game similar to rogue in format as a turn based game with a large map is always going to be slow to complete. In addition, the way that console input and output in the game is handled is also slow. Because display is already a class, and use of console output is discouraged, display could easily be replaced with a custom console window that supports clearing, direct keyboard input, and potentially multiple panes to separate turn events from the map for players.

Obviously this would take extra development time for unit staff, but the amount of time things like this could save for students would be immense. This is also a great opportunity for creative students to implement extra features such as viewing turn history, where the current UI doesn't inspire much change. I only improved mine because I got fed up with the amount of actions per turn. I was running out of hotkeys for things to test.

As an example - you could have a map pane, a turn pane, and a player menu pane which could be individually cleared.

(Also this isn't super related but the game really starts to crawl when there's a lot of tick() calls, like if you have a lot of grass on the map)

Skills Class/Skilled Interface

The ability to iterate over the skills of a Skilled Object would make it so much more useful. Currently you can only check one skill at a time using hasSkill, which leads to some pretty annoying code. You also can't get the skills list so there isn't a way to do a simple comparison of the skills of two objects - you have to check each skill one at a time. This can sometimes force you to use literals where you really shouldn't need to, which obviously limits the extensibility and maintainability of your code (or makes you ditch skills).

This could easily be changed by having a getSkills method that either returns the HashSet, which you could use Set.iterator() to iterate over (or have the student construct an ArrayList from it), or have it return an ArrayList constructed using the HashSet.
Although minor, it might also be useful to have a method that differentiates between inherent skills and skills obtained through holding at item, since currently hasSkill checks skills from inventory items as well as those that belong to the actor.

I guess the disadvantage would be that the skills list of an actor may now be accessible elsewhere, but that doesn't really seem like a problem. You can always return an unmodifiableList with the getter.

More Interfaces for Engine classes

Some of the interfaces in the Engine don't need to be in there, they could be moved to the interfaces package, specifically:

- Printable
- Skilled
- Weapon

Could be moved to allow students who wished to do so to edit them. This gives the advantage of more flexibility with some engine code.

Actions Class

Actions is a very limited Iterable that seemingly exists only to avoid null checks in the engine code. It caused some frustrations where it was generally much easier to use a List<Action> instead, due to the increased functionality of them, and convert them to and from Actions where necessary, which was fairly clunky to say the least. What if I wanted to use a stream, for example? In addition, currently it may be unclear whether a null check is required or not, since in engine there aren't any when Actions are being checked.

I know there's already a getUnmodifiableList method to give you a List<Action> if you want to do anything more with them, but then you have to convert that list back to actions if you've done anything.

It seems strange from my perspective that a non-nullable Iterable would be required here, anyway. Under what context would passing null into an actions list be an issue? Firstly, you probably shouldn't be passing null anyway, and 2, the receiving code could easily do a null check itself (which it often has to do anyway). It's also really easy to remove nulls from a List, if that was a genuine issue.

Although this would increase the number of nulls checks required in the code overall, I think the improved functionality and clarity outweigh that.

Minor Restrictions in the Engine

Some engine classes have seemingly arbitrary restrictions on them, including:

- Display printing only lets you print strings using println, and not print, for seemingly no reason. This could easily be changed by adding a method
- Items are either always allowed to be picked up and dropped or never allowed to be picked up and dropped. You'd have to override getAllowableActions to have an Item that could only be picked up by returning a unique pickup action, for example. While this is perfectly doable and not hard, it seems like an unnecessary restriction to simplify the portable attribute, that could easily be split into two booleans.
- The addExitFromHere method in GameMap is protected, when it doesn't really need to be. This would have made joining maps together at least look a lot nicer than it does now.

Documentation of Engine Code

I won't list them all, but there are plenty of places in engine where the documentation is simply lacking. For example:

- In Action the javadoc string for menuDescription simply reads "Returns a descriptive String" - for what purpose exactly? Is this only for outputting the result of an action? Is it only for the player? You have to do some extra digging to understand what it's actually for, when the addition of one extra sentence would have solved the problem. Not everyone knows how to use their IDE well and the extra time spent clicking through classes searching for methods instead of using IDE tools is frustrating
- Initially I had difficulty determining how direction worked in MoveAction.
- Very minor, and I'm sure it wasn't intended to be this way, but the "Go on, make a better one" in Menu reads more like a threat than an invitation.