Suggested Changes to the Engine

Skills

The first suggestion that would have assisted greatly during the program is a returnSkill method. This method could have returned an unmodifiable list of all skills that either the ground, item or actor has that could have been iterated through to see if that object had a skill. This would reduced the amount of classification that would was needed in classes such as LookForFood as instead of an if statement returning whether the dinosaur is a carnivore or a herbivore, it could have looked through the skills to see if either or both are there. In the current system a dinosaur would have to be assigned omnivore instead of just having both carnivore and herbivore skills. It would also have assisted in comparisons between objects, instead currently we have to find whether or not an object as a skill through the hasSkill method and then comparing that variable result in another hasSkill method for the other variable. If we were able to return a list of skills that an object had instead, the skill would not have to be hard coded based off whether true or false is returned from the hasSkill if statement, it could be easily passed straight in to the comparison with the next object. This would drastically reduce the amount of code required and increases the maintainability of the program for when new enums used for skills need to be added.

World

World class has two methods and these methods coupled with a lot of functionality and these can bring major problem if developer will like different world but would like to add in new features which could work the similar way but needed just a slight changes. Although the processActorTurn method of world is helpful in trying to make sure all actor has just about sufficient element to continue with game such as adding item action, do nothing action and etc before they can take turn to perform their play. A good way of ensuring an extendability of the world is to decoupled the processActorTurn to few method so that when World is extended by any subclass, they can simply added a few lines to it and simply call the decoupled method instead of copy pasting the line of code with new functionality. This adhere to DRY, where a method is decoupled to method and able to be reused every time without actually consist of the same line of code.

Movement Improvements

The second suggestion is based around the requirements from assignment 3. Because the Pteranodon was required to be capable of moving 2 spaces per turn, the movement had to be reworked, more specifically exit selection. This was never done in the engine as it is unneeded if everything only moves one square each turn as its only 1 line of code to return the exits from a location and for an actor to select one based off the behaviour that is being exercised. But this becomes more complex when the exits of the exits returned also need to be found. Because this requires several for loops to be able to return an immutable list of exits based off the number of moves that an actor can take, it is silly for that method to be placed into every behaviour that requires it. So therefore it should be placed in a place accessible by all behaviours for each separate actor.

The best position for this would be actor class in the engine as each actor would be able to return an immutable list of exits based off its current location and for however many moves that actor is allowed to move. We had to get around this by making a subclass under actor called GameCharacter that only had the method returnExits and was a superclass to all actor subclasses. We couldn't have just put the method in dinosaur as the trader subclasses may also have their movements changed.

The advantages of placing this new method into the actor function is that each actor is able to return and maintain its own possible exits rather than it being found in whichever behaviour needs to move an actor. This also reduces the amount of repeated code as its all housed in a central place that is accessible through any behaviour anyway. Another advantage of this would be that it would remove the GameCharacter subclass that we currently have to only house that single method which increases simplicity.