Assignment 3

Second iteration: `return(theJedi);`

In this assignment you will design and implement some quite complex new game functionality.

Project requirements

All the requirements stated in the Assignment 1 and 2 specifications still apply.

Make Reservoirs damageable

Modify `Reservoir` so that all reservoir objects start with 40 hitpoints.

- If the hitpoints of a reservoir fall below 20, its short description must change to "a damaged water reservoir", its long description must change to "a damaged water reservoir, leaking slowly", and its symbol must change to `V`.
- If the hitpoints of a reservoir fall to zero or below, its short description must change to "the wreckage of a water reservoir", its long description must change to "the wreckage of a water reservoir, surrounded by slightly damp soil", and its symbol must change to `X`.

Grenade

A grenade is an item that can be taken by an actor. After it is taken, it can be either put down again (with a leave command), or *thrown*. When a grenade is thrown, it explodes violently, doing damage to both actors and other entities, in "rings" of differing severity. It should cause damage as follows:

- Entities in the location where the grenade is thrown lose 20 hitpoints.
- Entities in locations that can be reached in one step from the location where the grenade is thrown lose 10 points.
- Entities in locations that can be reached in two steps from the location where the grenade is thrown lose 5 points.
- The actor that throws the grenade is not affected.

After a grenade is thrown, it is completely destroyed and disappears.

You must create a `Grenade` class with the functionality decribed above, and place several instances of Grenade at different locations in the world.

It might be a fun idea to test them by throwing them in or near the moisture farm once you have completed the `Reservoir` requirements above.

Jawa Sandcrawler

Jawas are small creatures that scavange the planet Tatooine for droids and other scrap material to use for trade. They travel in a vehicle called a "sandcrawler". Create a `Sandcrawler` with the following functionality:

- A sandcrawler moves in the same way as Ben Kenobi, but only moves every second turn.
- If a sandcrawler finds a droid in its location, the droid is taken inside the sandcrawler.
- A sandcrawler has a door that can be entered by any actor with force ability that is in the same location as the sandcrawler. When the actor enters the sandcrawler, the actor moves to the interior of the sandcrawler, which is a grid of at least four locations.[1]
  - All the droids taken by the sandcrawler must be in one of these interior locations.
  - One of the interior locations must have a door that can be user by any actor with force ability that is in that location. Exiting the door results in the actor being returned to the location in which the sandcrawler is located.

Design is important

One of the primary aims of this unit is for you to learn the fundamentals of object-oriented design. In order to get a high mark in this assignment, it will not be sufficient for your code to work. It must also adhere to the design principles covered in lectures, and in the required readings on Moodle.

If you would like informal feedback on your design at any time, please ask your lab demonstrator or come to a consultation session. Unit staff will be happy to discuss the quality of your design ideas with you.

Updating your design

You must document any design changes, including design extensions required to support the new functionality.

You will might find that some aspects of your existing design need to change to support the new functionality. You might also think of a better approach to some of the problems you've faced — your understanding of the problem will improve as you progress. You might also spot places where your existing code (and thus design) can benefit from refactoring. For any design changes that you make, please write a brief (one or two paragraph) explanation of your change and the reasons behind it.

If you have found that your original design documentation from Assignment 1 was too detailed to maintain, it's OK if you modify it to be less detailed and thus maintainable. You must still show all new classes and the relationships between them. It is not necessary to show all attributes and methods.

Coding and commenting standards

You must adhere to the Java coding standards that were posted on Moodle earlier in the semester (Week 4).

Write javadoc comments for *at least* all public methods and attributes in your classes. If you change any methods in the supplied codebase, please take care to keep their documentation up to date.

You will be marked on your adherence to the standards, javadoc, and general commenting guidelines that were posted to Moodle earlier in the semester.

[1] Yes, OK, I guess this makes it rather like a TARDIS :)

Bonus marks

As stated in Assignment 2, there are bonus marks available for groups who come up with ideas for new features for the game, and then design and implement them. These could be new kinds of actor or entity, with new behaviours. They could be new kinds of location — perhaps even other worlds! They must be different from any features that have been requirements in FIT2099 in previous semesters.

A feature must be quite complex to qualify for a bonus mark. You must discuss your plans for bonus mark features with your tutor and gain approval before beginning work on them. Bonus marks will be awarded after the submission of Assignment 3.

A maximum of three bonus marks are available (these are whole marks for the unit). No one has to attempt bonus marks, and it is possible to get full marks for the unit without any bonus marks.

Submission instructions

The assignment is to be checked in using the GitLab repository that was created for your group.

You must agree on a work breakdown for the new work in Assignment 3 and update your Work Breakdown Agreement to cover it. Do this as soon as possible, and make sure each team member accepts it. We will use the agreement and Git commit histories to help resolve any disputes between teammates about work allocation, quality, and timeliness.

The due date for this assignment is *Sunday 27th May at 23:59*. We will mark your Assignment 3 on the state of the "master" branch of your GitLab repository at that time. If you've done any work in other branches, make sure you merge them back into master before the due time.

Unless a team member has applied for and received special consideration according to the Monash Special Consideration Policy,[2] late submissions will be penalized at 10% per day late.

It is both team members' responsibility to ensure that the correct versions of the documentation and code are present in the repository by the due date and time. Once both teammates have agreed on a final Assignment 3 submission, do not make further commits to the master branch of the repository until the due date has passed, without the agreement of your teammate. If you want to continue to make changes to the repository for some reason, make another branch.

We will take your Work Breakdown Agreement into account when marking if there seems to be a major discrepancy in the quality of different parts of the submission, or if the code is missing major sections. Students whose work is inadequate in either quality or quantity will be penalized, and their partners will be compensated. If you choose to reallocate tasks, make sure you keep your WBA up to date.

[2] `http://www.monash.edu/exams/changes/special-consideration`

Marking Criteria

This assignment will be marked on:

- Functional completeness
- Adherence to design principles
- Code quality
  - readability
  - adherence to Java coding standards
  - quality of comments
  - maintainability
- Quality of updated documentation, if any
- Correct use of GitLab

Marks may also be **deducted** for:

- late submission
- inadequate individual contribution to the project
- academic integrity breaches