Assignment 3

Goonmaker

Due: Friday 31st May, 23:55
For learning outcomes, see final page of this spec

In this assignment you will design and implement some new game functionality, and write a set of recommendations for changes to the game engine.

Project requirements

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

You must document your designs as you did for Assignment 1 and 2, although this time you do not need to create a design rationale.

Going to the Moon

As of the end of Assignment 2, it was possible for the player to build a rocket, but there wasn't anything they could do with it. We are changing that now: the rocket will be able to take the player to a secret moonbase.

- When the player is standing at the assembled rocket, they must have the option to fly to the moonbase.
- The moonbase will be a separate map, at least 10 $\times$ 10 squares in size.
- The moonbase is populated with the same types of enemies as the lair on Earth: Goons, Grunts, and/or Ninjas
- The rocket must also appear in the moonbase map, with an option to fly back to the lair on Earth. When the player moves between maps, they must appear at the rocket's location, to give the impression that the rocket is moving.

Spacesuit and oxygen

In order to move around in the moonbase, the player must be wearing a spacesuit and have an adequate oxygen supply.

Add a spacesuit item to the Earth lair map. Also add an oxygen dispenser. The oxygen dispenser has a button that, when pressed, causes the dispenser to produce an oxygen tank in its location on the **next turn**. The button does not work while the dispenser is producing the tank, or while there is an oxygen tank in the location.

Every turn spent in the moonbase costs the player one point of oxygen. A new oxygen tank holds 10 points worth of oxygen. If the player runs out of oxygen on the moon, a safety system automatically transports them back to the rocket's location on Earth.

The enemies on the moonbase don't need extra oxygen because they have cybernetic implants that provide their bodies with energy.

Final boss fight

The games' final boss is Yugo Maxx, evil millionaire industrialist. When not standing next to the player, he wanders the map at random. He is planning to blow up the moon so that he can extract its minerals – he must be stopped! Unfortunately, he hits as hard as a Goon, and he is wearing an exoskeleton that makes him invulnerable to damage. The exoskeleton must be destroyed before Maxx can be brought to justice.

There is only one substance in the universe that can take out that exoskeleton: water.

- Add Yugo Maxx to the map, at least 10 squares away from the rocket.
- Add a water pistol item to the moon map, at least 5 squares away from the rocket. The water pistol starts out empty.
- Add a pool of water to the Earth map. You can't walk on water, but when you're standing next to it you can fill an empty water pistol.
- Squirting water from a full water pistol onto Yugo Maxx has a 70% chance of destroying his exoskeleton, making him vulnerable. It empties the water pistol whether it hits or misses.

Ending the game

At the moment, there is no way to end the game, and if the player is knocked out then the game crashes.

You must add the following endings:

- A "quit game" option in the menu
- A "player loses" ending for when the player is knocked out
- A "player wins" ending for when the player defeats Yugo Maxx and carries his unconscious body back to Earth

Recommendations for change to the game engine

Over the course of your three assignments, you have had to become familiar with the game engine (located in the package `edu.monash.fit2099.engine`). You have almost certainly found aspects of the game engine design and implementation difficult to understand, or frustrating to use.

Write a short document (e.g. two or three A4 pages of text) describing changes you would recommend for the game engine. For each change you recommend, you must explain:

- what problem you perceive
- the design change you propose to address the problem
- the advantages, and any disadvantages, of the proposed change

If you use UML diagrams, don't count the space they take up in the "two or three A4 pages of text".

Note that any recommendations you make must be suitable for any application of the game engine. They must not be specific to the current scenario – we change that every year. (Last year it was Harry Potter, and the year before that it was Star Wars.)

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 consult your lab demonstrator in a lab or attend any consultation session.

Updating your design

You 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 requirements you have already implemented — your understanding of the requirements and codebase will have improved as you have progressed. You might also spot places where your existing code (and thus design) can benefit from refactoring.

If you want to update your design, you may do so; if you decide to do this, be sure to update your design documents so that they match the code, and write a brief explanation of your changes and the reasons behind them. his will help your marker understand the thinking behind your code.

Coding and commenting standards

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

Write javadoc comments for *at least* all public methods and attributes in your classes.

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

Bonus marks

As stated in the Assignment 2 specification, there are bonus marks available for groups that come up with ideas for new features for the game, and then design and implement them.

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. Different tutors may have different standards so to ensure that bonuses are treated fairly, they will be determined centrally by an independent assessor. 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 due date for this assignment is *Friday 31st May at 23:55*, your local time. We will mark your Assignment 3 on the state of the "master" branch of your Monash GitLab repository at that time. If you have done any work in other branches, make sure you merge it into master before the due time.

Do not create a new copy of your work for Assignment 3. Continue working on the same files, in the same directory structure (you might like to add a tag to the your final Assignment 2 commit before starting on Assignment 3, so you can find that version easily).[1]

As we said above, you may update your design and implementation if you find that your Assignment 2 solution is not suitable for extension, or if you think of a better approach during Assignment 3.

You must update your Work Breakdown Agreement to include the work necessary for the Assignment 3 requirements, and any bonus mark work you attempt.

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.

Marking Criteria

This assignment will be marked on:

- Functional completeness
- Design quality
  - adherence to design principles discussed in lectures
  - UML notation consistency and appropriateness

[1] `https://git-scm.com/book/en/v2/Git-Basics-Tagging`
[2] `http://www.monash.edu/exams/changes/special-consideration`

  - consistency between design artefacts
  - clarity of writing.
  - level of detail (this should be sufficient but not overwhelming)
- Code quality
  - readability
  - adherence to Java coding standards
  - quality of comments
  - maintainability (application of the principles discussed in lectures)
- Correct use of GitLab
- Quality of recommendations for changes to the codebase
  - understanding of codebase displayed
  - usefulness of recommendations
  - understandability of recommendations
  - generality of recommendations (i.e. must not be specific to the current scenario)
  - practicality of recommendations
  - clarity of writing

Marks may also be **deducted** for:

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

Note: Learning outcomes for this assignment

This assignment is intended to develop and assess the following unit learning outcomes:

1. Iteratively construct object-oriented designs for small to medium-size software systems, and describe these designs using standard software engineering notations including UML class diagrams (in conceptual and concrete forms), UML interaction diagrams and, if applicable, UML state diagrams;

3. Implement object-oriented designs in an object-oriented programming language such as Java, using object-oriented programming constructs such as classes, inheritance, abstract classes, and generics as appropriate;

5. Use software engineering tools including UML drawing tools, integrated development environments, and revision control to create, edit, and manage artifacts created during the development process.

To demonstrate your ability, you will be expected to:

- design and implement further extensions to the system
- use an integrated development environment to do so
- update your UML class diagrams and interaction diagrams as required, to ensure that they match your implementation
- comment on the design of the existing game engine
- use git to manage your team's files and documents

The marking scheme for this assignment reflects these expectations.