Assignment 1: Planning and Design

Live And Let Design

Due: Thursday 18 April at 11:55pm, your local time

For the rest of the semester, you will be working in teams on a relatively large software project. You will design and implement new functionality to add to an existing system that we will provide to you.

This assignment has been designed to be done in pairs, and we will not allow you to work on your own. You will be paired with another student in your lab by unit staff. If there is an odd number of students in your lab, your lab may have *one* team of three — the teaching staff at your campus will decide which team.

To help you manage your time on this project, we have divided it into three phases. These will be assessed separately. In this phase you will familiarise yourselves with the code base, decide on how to split up the tasks for the next assignment, and most importantly, create some preliminary designs for the extra functionality you will implement in the next phase (Assignment 2). You will extend the system again in Assignment 3, so do the best you can to keep your design and implementation for the system extensible and maintainable.

Managing and Submitting Your Work

This is a large project, and you will need a way to share files with your partner and unit staff. Instead of requiring you to submit your work on Moodle, we will provide you with a Monash-hosted GitLab repository that we can also read. There are a number of advantages to doing it this way:

- You learn to use Git to manage code and other software engineering artefacts. Git is a fully-featured modern version control system. It is the most widely used version control software for commercial, open-source, and hobbyist software projects today.
- It provides you with a mechanism for sharing files with your partner. You will be able to access your Monash GitLab repository from Monash and from home — anywhere you have internet access. Note that Git support is built into Eclipse, and almost all modern IDEs.
- You do not need to "submit" anything. Instead, ww will use the state of your Git repository at the due date and time. You just need to ensure that the master branch is ready for marking at that time.
- Your changes are automatically tracked. If you and your partner have a dispute about sharing the workload, unit staff can easily see whether you are adhering to your agreed tasks and timelines.
- Every time you make a change, you include a comment that summarises what was done. This greatly aids communication within your team.

Getting Started

The initial code base is available in a repository that has been created for you on `git.infotech.monash.edu`. It includes a folder containing design documents for the system. Download it and familiarize yourself with the code and its documentation.

Background

You will be working on a text-based "rogue-like" game. Rogue-like games are named after the first such program: a fantasy game named `rogue`. They were very popular in the days before home computers were capable of elaborate graphics, and still have a small but dedicated following. If you would like more information about roguelike games, a good site is `http://www.roguebasin.com/`.

The theme of this game is related to the work you have done in the labs, but instead of running an evil genius's lair, the user will play as a good guy who is invading the lair to stop the bad guy's evil plan. This game is inspired by movies such as the James Bond series, and we're trying to keep it suitable for children by limiting the amount of violence involved. Enemies should be knocked out, chased away, or captured rather than killed or destroyed.

As it stands, the game functions, but is missing a lot of desired functionality. Over the course of this project, you will incrementally add new features to the game.

What to submit for Assignment 1

You are expected to produce the following three design artefacts in Assignment 1:

- Class diagrams
- Interaction diagrams
- A design rationale

You will be implementing your design later, but for Assignment 1 you are not required to write any code. Instead, you must produce preliminary *design documentation* to explain how you are going to add the specified new functionality to the system. The new functionality is specified in the **Project Requirements** section.

We expect that you will produce *class diagrams* for all of the new features. Your class diagrams do not have to show the entire system. You only need to show the new parts, the parts that you expect to change, and enough information to let readers know where your new classes fit into the existing system. As it is likely that the precise names and signatures of methods will be refactored during development, you do not have to put them in this class diagram. However, the overall responsibilities of the class need to be documented *somewhere*, as you will need this information to be able to begin implementation.

To help us understand how your system will work, you must also write a *design rationale* to explain the choices you made. You must explain both how your proposed system will work and *why* you chose to do it that way.

You should use *interaction diagrams* (e.g. sequence or collaboration diagrams) to show how objects interact within your system. These are only needed for complex interactions. If you are unsure if an interaction is complex enough to require an explanatory diagram, consult your tutor.

The design (which includes *all* the diagrams and text that you create) must clearly show:

- what classes will exist in your extended system
- what the roles and responsibilities of any new or significantly modified classes are
- how these classes relate to and interact with the existing system
- how the (existing and new) classes will interact to deliver the required functionality

You are not required to create new documentation for components of the existing system that you *do not* plan to change.

Your design documents may be created with any tool that you choose — including a pen and paper if you can do so legibly. However, **you must create PDF, JPEG or PNG images of your design documents**

in your Git repository. We can not guarantee that your marker will have the same tools available that you used (or the same versions).

If you want a UML diagramming tool, UMLet (`http://www.umlet.com`) is a free, easy-to-use UML diagramming tool that is particularly suitable for beginners. The diagrams in the initial code base were created using UMlet, and the source files are included. There is an online version of UMLet at `http://www.umletino.com` that you can use on any computer that has JavaScript enabled.

As you work on your design, you must store it in your Git repository.

Git resources and policy

You are required to use Git to manage all project artefacts: code *and documents*. You have already starting using Git in your lab exercises. See the resources provided there and on Moodle for information on using Git. As in your lab tasks, a Git repository will be provided for you on the Monash GitLab server. **You must use the repository provided. Do not create a new repository**. We will use your Git commit log to see who has been contributing to the project and who has not. That means that it is *very important* that you commit and push your work frequently — *including your design documentation*.

Storing project artifacts elsewhere and commiting them to your repository just before the due date is **not** acceptable and will be penalized as not complying with the assignment specification. It is also highly risky — laptops get lost, hard disks become corrupt, and cloud services go offline at crucial moments (if the Monash GitLab server goes down, it is our problem, not yours).

Your team may adopt any policy you choose for branching and merging your repository; however, you will be marked on your master branch. If you have not integrated your changes to master by the due date and time, they will not be marked and you will receive zero marks for that functionality.

Work Breakdown Agreement

We require you to create a simple Work Breakdown Agreement (WBA) to let us know how you plan to divide the work between members of your team.

Your WBA must explain:

- who will be responsible for *producing* each deliverable (whether it is a part of the system, an internal document, or an externally-deliverable document),
- who will be responsible for *testing or reviewing* each deliverable, and
- the dates by which the deliverable, test, or review needs to be completed.

Both partners must accept this document. You can do this by adding your name to the WBA and pushing it to the server with a commit comment that says "I accept this Work Breakdown Agreement". (Rest assured that if the WBA is changed after this time, we will be able to tell!)

We do not require you to follow a template for the WBA. A simple `.doc`, `.md`, or `.txt` file that contains the information we need will be sufficient. This document is not worth marks in itself, but it is a hurdle requirement: your submission will not be accepted if there is no WBA. We will use it to make sure that you are allocating work fairly, and to make sure that we can hold individuals responsible for their contributions to the team.

We will use this document for accountability. If a team member complains that their partner is not contributing adequately to the project, or that their contributions are not being made in time to allow for review or debugging, FIT2099 staff will look at the partner's contribution in the logs. If we decide that the complaint is justified, this will be taken into account in the marking: students who do not contribute will be penalized and may fail, while students whose partner is not contributing adequately may have their mark scaled so as not to count any functionality that is missing due to the actions of their teammate.

Assignment 1 and 2 Requirements

You will be provided with a codebase for a text-based game. This game is based on an engine (located in the packages with prefix `edu.monash.fit2099`) that has been written for you to use. We plan to reuse this engine in future years, to support games based on many different pop-culture themes. A simple set of packages that uses this engine for a game set in our fictional universe has been created.

We have created a set of requirements that you must add to the system. If you are worried about the interpretation of these requirements, we encourage you to post a question on the unit Moodle Discussion Forum. Teaching staff will be quick to respond, and other students will also benefit from seeing the discussion. Always look on the forum before you post in case somebody has already posted a similar question!

Note that you are only expected to *design a solution* that can support these requirements for this initial Assignment. You will implement this solution (and refactor it if necessary) in Assignment 2. Assignment 3 will bring a new set of requirements for you to add to the system.

Doors and keys

At the moment, the map consists of walls (which you can't pass through) and ground (which you can pass through). You must add doors to this system.

A door can't be passed through until it has been opened with a key. A key is an item that is dropped by enemies when they have been knocked out.

New types of enemy

Currently, there is only one type of enemy: the Grunt. Grunts can follow the player around and slap them, but don't do much damage. We want you to add two new types of enemy: Goons and Ninjas.

- **Goons** also follow the player, but do twice as much damage as Grunts. They also have a 10% chance *on each turn* of shouting an insult at the player. You can come up with your own list of insults for the Goon to shout (but remember our target audience and keep the language suitable for children).
- **Ninjas** stay in one place unless they can see that the player is within 5 squares of them. Then they will do the following:
  - Throw a bag of stun powder at the player. This has a 50% chance of hitting, and if it hits (and the player is not already stunned), will stun the player for two turns. While stunned, the player will be unable to successfully perform any actions other than waiting. If the player is already stunned, the stun powder has no effect.
  - Move one space away from the player.

Q

Q is a non-player character (NPC) – that is, an Actor who is friendly to the player, or at least not hostile. In the James Bond movies, Q provides the hero with cool technological gadgets. In our game, Q will supply the player with part of the rocket.

Q should support the following actions:

- **Give plans**. If the player is holding rocket plans, they should be able to give them to Q. This will cause the plans to be removed and replaced by the rocket body. Q will then disappear with a cheery wave.
- **Talk**. Talking to Q *without* rocket plans in the player's inventory should result in Q saying "I can give you something that will help, but I'm going to need the plans." If the player does have the rocket plans in their inventory, Q should say "Hand them over, I don't have all day!"

Q should wander around the map at random.

Miniboss: Doctor Maybe

Doctor Maybe is a researcher rather than a combat specialist, so isn't very tough: he has half the hit points and does half the damage of a Grunt, and does not move at all. He should be placed inside a locked room.

When defeated, Doctor Maybe must drop a rocket engine.

Building a rocket

The player's goal is to build a rocket. This will involve three new items:

- The **rocket plans**. Initially, these should be inside a room with a locked door.
- The **rocket body**. Q will give this to the player in return for the rocket plans.
- The **rocket engine**. This is dropped when Doctor Maybe is defeated.

Add these three items to the game. You must also add a special location to the map: **the rocket pad**. To build the rocket, the player needs to place both the rocket body and the rocket engine on the rocket pad.

Infrastructure

The engine packages are the way they are for good reasons. Unless a task specifically asks you to do so, **do not modify the engine code!** Doing this will make the game engine much less reusable — and cost you marks.

Starting Coding

You are **not required** to do any coding for Assignment 1, but you are free to start coding whenever you like. Doing so may help you to understand the existing code base better and let you test the feasibility of your design ideas. No new or changed code in your repository will affect your mark for Assignment 1 — we're not even going to look at it until Assignment 2.

Submission instructions

You must put your design documents and work breakdown agreement (in one of the acceptable file formats listed earlier) in the `design-docs` folder of your Monash GitLab repository.

The due date for this assignment is **Thursday 18 April at 11:55pm, your local time**. Please create a tag named **Assignment 1** to show the version we should mark. You can do this in the web interface by opening the Repository submenu in the left sidebar, clicking Tags, and then clicking the green New Tag button. Create the tag from your `master` branch. Before you do this, **please make sure that your master branch is up to date!**

If no tag exists, we will mark a snapshot of your repository as it was at the due time. This means that you will need to notify your marker if you finished late and let them know which version they should check out.

Unless a team member has applied for and received special consideration according to the Monash Special Consideration Policy,[1] late submissions will be penalized at 10% per working day late.

It is both team members' responsibility to ensure that the correct versions of the documentation are present in the repository by the due date and time. Once both teammates have agreed on a final Assignment 1 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 or create a tag as described above.

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

How you will be marked

After assessing your submission, your tutor will email you *individually* 24 hours before your scheduled week 8 lab and ask you each a question about an aspect of your design. You must write a response to the tutor's emailed question in your own words, *without discussing your answer with your teammate*. This answer must be placed in a file and committed to the repository before the beginning of your lab. This response should be no longer than 250 words.

Completing the response to the question is a hurdle for receiving a mark for assignment 1.

Marking Criteria

This assignment will be marked on:

Design completeness Does your design support the functionality we have specified?

Design quality Does your design take into account the programming and design principles we have discussed in lectures? Look for the principles like

Do Not Repeat Yourself

Practicality Can your design be implemented as it is described in your submission?

Following the brief Does your design comply with the constraints we have placed upon it — for instance, does your design leave the engine untouched, as required?

Documentation quality Does your design documentation clearly communicate your proposed changes to the system? This can include:

- UML notation consistency and appropriateness
- consistency between artefacts
- clarity of writing
- level of detail (this should be sufficient but not overwhelming)

Explanation Can you adequately explain your design and the reasoning behind it, both in your rationale and in response to the question from your marker?

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;

2. Evaluate the quality of object-oriented software designs, both in terms of meeting user requirements and in terms of good design principles, using appropriate domain vocabulary to do so;

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:

- read and understand UML design documentation for an existing Java system
- propose a design for additional functionality for this system
- create UML class diagrams and interaction diagrams to document your design, using a UML drawing tool such as UMLet or Visual Paradigm – you are free to choose which one
- write a design rationale evaluating your proposed design and outlining some alternatives
- use git to manage your team's files and documents

The marking scheme for this assignment reflects these expectations.