Assignment 1: Planning and Design

Star Wars Episode IV: A New Project

For the remainder of the semester, you will be working in pairs 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's 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'll familiarise yourselves with the code base, decide on how to split up the tasks for the next assignment, and most importantly, make some preliminary designs for the extra functionality you'll 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'll 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:

- It provides you with a mechanism for sharing files with your partner. You'll be able to access your GitLab repository from Monash and from home — anywhere you've got internet access. Note that `git` support is built into Eclipse, and almost all modern IDEs.
- It uses a fully-featured modern version control system — `git` — that is widely used for commercial, open-source, and hobbyist software projects.
- You don't need to "submit" anything. Instead, we'll check out your master branch at the due date — no need for you to do anything special.
- Your changes are automatically tracked. If you and your partner have a dispute about sharing the workload, unit staff can easily see whether you're 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 zip archive on Moodle. This archive includes a directory with some UML design documents that describe the code base.

Download the archive and create a project in your IDE so that you can explore it. You will need to spend considerable time trying out its functionality, and reading the code in the `starwars.*` packages.

Once you have your GitLab repository, you will add this code base to it, and commit all changes to the repository.

Assignment 1 Tasks

You will be working on a text-based "rogue-like" game set in the Star Wars universe. 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'd like more information about roguelike games, a good site is `http://www.roguebasin.com/`.

If you are not familiar with Star Wars, you may wish to look it up on IMDB. The first Star Wars movie to be produced was Episode IV: A New Hope (`http://www.imdb.com/title/tt0076759/`). A very comprehensive fan site is Wookieepedia, which you will find at `http://starwars.wikia.com/`.

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.

Design Documents

For Assignment 1, you are not required to write any code. Instead, you must produce preliminary *design documentation* to explain how you're going to add the specified new functionality to the system. The new functionality is specified **Project Requirements** section.

We expect that you will produce *class diagrams* for all of the new features. Your class diagrams don't 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 don't have to put them in this class diagram. However, the overall responsibilities of the class need to be documented *somehwere*, as you'll 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 exist in your extended system.
- what the role 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 *don't* 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 upload PDF, JPEG or PNG images of your design documents**. We can't 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.

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

`git` resources and policy

You are required to use `git` for version control. You will find introductory information on how `git` works at `https://git-scm.com/doc` — view the videos at the bottom of the page in order to help you understand how to manage your repository. You can read about using GitLab at `https://gitlab.com/help/#getting-started-with-gitlab`. (Note: you will be using repositories hosted at Monash (`https://git.infotech.monash.edu/`), not `https://gitlab.com`.)

Your code *and documents* will be stored on GitLab. We will use your project's commit log to work out who's 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*. It's also good practice to do this as it makes it much less likely that you'll lose work to a crash or accidentally deleting it.

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 submission instructions. It's also highly risky — laptops get lost, hard disks become corrupt, and cloud services go offline at crucial moments (if GitLab goes down, it's our problem, not yours).

Your team may adopt any policy you choose for branching and merging your repository; however, you'll be marked on your master branch. If you haven't integrated your changes to master by the due date and time, there's a strong chance that they won't be marked and you will receive zero marks for that functionality.

Work Breakdown Agreement

We also require you to submit a very 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's 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 don't require you to follow a template for the WBA. A simple `.doc` or `.txt` file that contains the information we need will be sufficient. This document isn't worth marks in itself, but it is a hurdle requirement: your submission won't be accepted if there is no WBA. We will use it to make sure that you're 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 isn't contributing adequately to the project, or that their contributions aren't 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.

Project requirements

You have been provided with a large codebase for a text-based game set in the Star Wars universe. This game is based on an engine (located in the packages with prefix `edu.monash.fit2099`)that has been used for several years for games based in many different pop-culture universes, but the Star Wars code that lies on top of it is new.

We have created a set of requirements that you must add to the Star Wars system. If you're worried about the interpretation of these requirements, we encourage you to post a question on Moodle. Teaching staff will be quick to respond, and other students will also benefit from seeing the discussion. Of course, it's a good idea to look in the forum before you post in case somebody's 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'll be implementing this solution (and refactoring it if necessary) in Assignment 2. Assignment 3 will bring a new set of brainstormed requirements for you to add to the system.

Leave Affordance

Some entities have a `Take` affordance (e.g. lightsabres, blasters). This allows an actor to pick the item up. There is currently no way for an actor to put down the item the actor is holding. This means that the actor cannot pick up anything else.

Add a `Leave` affordance that allows an actor to put down the object they are currently carrying. After this is done, the actor should be holding nothing, the item should be in the location of the actor when it was done, and the item should be able to be picked up again.

Force ability

Some, but not all, people[1] (including Ben and Luke) have the ability to use the Force.

Lightsabres

Anybody can pick a lightsabre up, but only people with a lot of Force ability can wield one and use it as a weapon.

Ben Kenobi

Ben can train Luke when they are in the same location. Training Luke has the effect of raising his force ability to the extent that Luke can wield a lightsabre.

Droids

Droids can't use the Force.

A droid can have another actor as its owner. If a droid has an owner:

- if a it is in the same location as its owner, it will stay there. If the droid's owner is in a neighbouring location, it will move to that location.
- if a droid cannot find its owner in its current or neighbouring locations, it will pick a direction at random and move in that direction. It will keep moving in that direction until it finds its owner, or can no longer move in that direction. If it can no longer move in its current direction, it will pick a new direction at random and move in that direction.

If a droid has no owner, it will not move.

Droids lose health when they move in Badlands.

Droids don't die, but they become immobile when their health runs out.

[1] This is Star Wars, so the word "people" includes aliens.

Infrastructure

The engine packages are the way they are for good reasons. **Don't 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**. You are free to start coding as soon as you like though. Indeed doing so may help you to understand the existing code base better, and the feasibility of your design ideas. No new or changed code in your repository will affect your mark for Assignment 1 (or even be looked at during marking).

Submission instructions

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

The due date for this assignment is *Sunday 29th April at 23:59*. Note that this a week later than it says in the unit guide. We wish to give you some more time due to recent disruption of some labs by Good Friday, etc. Due dates for the later assignments will be adjusted accordingly. We will mark your Assignment 1 based on the state of the "master" branch of your GitLab repository at that 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 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.

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 place 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 artifacts.
- clarity of writing.
- level of detail (this should be sufficient but not overwhelming)

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