FIT2099 Assignment Rules

Semester 2, 2019

For the second half 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. These are the rules that apply to this project.

Working in a team

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.

If you tell us who you would like to work with, we will try to accommodate you, but we can't promise that everybody will have first pick of partners. We also can't promise that you will get to work in a group of three, even if you're the first team to ask your marker to do so. Please be aware that we might need to alter the composition of your team during semester – if a student in your class drops out, we'll need to find a new team for their partner.

Work Breakdown Agreements

Before you begin each Assignment, 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.

Here is a step-by-step guide to the WBA.

1. Discuss the WBA with your team. Decide on how you will break down the tasks. This must be listed in enough detail that your marker will know who to penalize if something doesn't get done.
2. Write the WBA. Your WBA must explain:
   - who will be responsible for *producing* each deliverable (whether it is a part of the system, something that is needed within your team, 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.

   Push the WBA to your repository.
3. **Each team member, one at a time,** must then:
   (a) pull the project (to get the latest version of the WBA)
   (b) append the line "I accept this WBA" to the WBA
   (c) commit the change (with a comment such as "Accepted the WBA")
   (d) push your commit to the server

   This needs to be done one at a time so that you don't overwrite your partner's acceptance with your own, because although git is good at merging changes, that only works with text files. It doesn't work on PDFs or .docx files.
4. After the WBA has been accepted, don't change it. That would be like changing a contract after it had been signed: it's against the rules, and it's unethical. If you discover a problem with it, speak to your marker.

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 correctly signed 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. **It is a good idea to show your WBA to your marker at the start of the project to make sure that it is detailed enough.**

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.

Using git

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, we 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.

We *require* all project artefacts to be pushed to the repository created for you on `https://git.infotech.monash.edu`. Learning resources have been provided for you on Moodle and during bootcamps to help you understand how to use git and how to fix problems when they arise.

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). We *suggest* that you commit your work to your local repository very frequently, and push it to the server as soon as you can get the code to run without breaking any existing functionality. Frequent commits make it much less likely that you will lose any of your work, and frequent pushes (and pulls) make merge conflicts less likely.

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.

Coping with server outages

Although git is an efficient version control system, the Monash server has occasionally failed to cope with peak load. This causes slowdowns and outages. To deal with this, try the following tips:

- Don't use the web UI for assignment submissions. Instead, use git via the plugins in Eclipse or IntelliJ, a third-party client such as GitKraken or TortoiseGit, or the command-line client. The back end is often responsive even when the web client is hanging.
- Push early and often. The longer you leave it between pushes, the more data will need to be transferred, and the more likely it is that an outage will affect you.
- Make sure your classfiles (and anything else you don't need to share) are in `.gitignore`. This also helps minimize the data that needs to be transferred.
- Use text formats rather than binary formats wherever possible. That means use .txt or .md rather than Word for text documents – there's nothing in FIT2099 that requires sophisticated formatting anyway. When you modify a binary file, git will transfer the whole thing, but if you modify a text file it will transfer a short description of how the file needs to be changed.
- Wait a little while, then try to push again (without committing). As long as the push doesn't happen suspiciously late, we'll be looking at the timestamp on the commit, not the timestamp on the push. As long as you've committed before the due time, it's okay if it takes a little while to complete the push.

About roguelike games

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/`.

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. You will be expected to design some new features in Assignment 1, then implement them in Assignment 2. In Assignment 3, you will both design and implement another set of features.

Restrictions on your project

You are not allowed to change game engine code. The "game engine" is the code in the package `edu.monash.fit2099.engine`.

You *are* allowed to do any of the following:

- create new classes (enums, interfaces, etc.) outside the engine
- modify existing classes that are outside the engine
- call public methods on instances of engine classes
- instantiate interfaces defined in the engine (if they're public)
- inherit from classes defined in the engine, and use their protected and public methods

Documenting your designs

You are expected to be able to produce *UML class diagrams* and *UML interaction diagrams* (i.e. sequence diagrams and/or collaboration diagrams). Class diagrams were presented during the live coding sessions in the first five weeks of semester. Interaction diagrams are covered in lectures in Week 6. You have been provided with a set of class diagrams that cover the engine code and basic game code that we have provided to you.

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**[1]. 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. 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. This will help your marker verify that you have been completing your share of the work on time as agreed.

Class diagrams

We expect that you will produce *class diagrams* that cover new or changed classes that implement 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. This can be done in a separate text document, or you can describe the responsibilities of each class in its classbox in the class diagram as recommended by Fowler[2].

When you are drawing your class diagram, please bear in mind that it has a purpose: it's there to help your marker understand your system, and to evaluate the quality of your design. Your marker will want to know:

- what things you have chosen to model using classes, and how those classes interact
- whether you have considered how desired functionality can be implemented
- what the roles and responsibilities of any new or significantly modified classes are
- how these classes relate to and interact with the existing system
- whether you are using new object-oriented techniques such as inheritance in an appropriate way
- whether your classes, methods, and packages are too large or small
- whether you are trying to minimize dependencies between classes, methods, and packages.

You don't need to put all of your classes into one class diagram. In fact, doing so would probably make the diagram very hard to read. Instead, consider drawing separate class diagrams for each package. You should show all classes, dependencies, and associations within the package, but the only classes in other packages that need to be shown are those that your classes depend on. See the game package documentation for an example of this style of documentation.

Interaction diagrams

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. As a rule of thumb, a "complex interaction" is one which:

- involves at least three interacting classes, and
- involves more than one class sending messages.

If you are unsure if an interaction is complex enough to require an explanatory diagram, consult your marker.

[1] Also include your original source files, so that you can share them with your partner and so that you've got a backup of them.
[2] *UML Distilled*, 3rd edition

Your interaction diagrams should show:

- all objects involved in the interaction, and their classes
- all messages sent (i.e. methods called)
- values returned from messages, if not trivial

Scope of an interaction diagram

In other units, especially in domains such as business process analysis, it is common for an interaction diagram to be drawn for a *use case*. The UML itself does not limit the use of diagrams in this way. For the purposes of this unit, use-case based documentation probably involve a lot of boilerplate documentation showing the user selecting menu selections. For that reason, we suggest that you limit the scope of your interaction diagrams to showing the interactions behind one or two key method calls each – you can (and should!) place a note or label each diagram to make its scope clear.

Good candidates for interaction diagrams often include complex `playTurn()` or `Behaviour` operations, that need to take many factors into account when choosing which `Action` an `Actor` will take on the next turn. However, you can diagram out any operation that you feel is complicated enough to be explained in detail.

When to start coding

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.

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. Any new or changed code in your repository will not affect your mark for Assignment 1 — we're not even going to look at it until Assignment 2.

Submission instructions

Unless a team member has applied for and received special consideration according to the Monash Special Consideration Policy,[3] 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 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.

How you will be marked

As well as looking at your submission offline, your marker will interview each team member during class time in the week after the assignment is due. Team members are expected to be able to answer questions about *any* part of the design or codebase, not just the part they completed – you need to make sure that communication within your team is good enough that everybody understands what is going on.

Team members who don't seem to understand the submission will lose marks on the interview component, and may also lose further marks if evidence suggests that they have not made a sufficient contribution to the project.

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