Assignment Rules (+ Survival Kit)

Learning Outcomes & Materials

These assignments are intended to develop and assess the following unit learning outcomes:

- **LO1.** 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;
- **LO2.** 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; *(Assignment 1)*
- **LO3**. Implement object-oriented designs in an object-oriented programming language (i.e., Java), using object-oriented programming constructs such as classes, inheritance, abstract classes, and generics as appropriate; *(Assignment 2 and 3)*
- **LO4.** Use available language tools, such as debuggers and profilers, and good programming practice to debug their implementations systematically and efficiently. *(Assignment 2 and 3)*
- **LO5**. 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 will reflect these expectations

Learning Materials

Please download this .zip file to familiarise yourself with the game engine, documentation, and more. The base code will be automatically available in your group's repository (Gitlab). We intentionally make this `.zip` file below doesn't have a `game` package. We don't want you to write code straightaway.

fit2099-assignment.zip

> Repeat this mantra: Design, write code, test, fix design, fix code, repeat

Note: You **must NOT follow** demo apps' design decisions; they only show how to use the engine **NOT** how to design a proper system with object-oriented principles.

NOTE: This module is not an assignment instruction.

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 artefacts elsewhere and committing them to your repository just before the due date is **NOT acceptable** and will be penalised for 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.

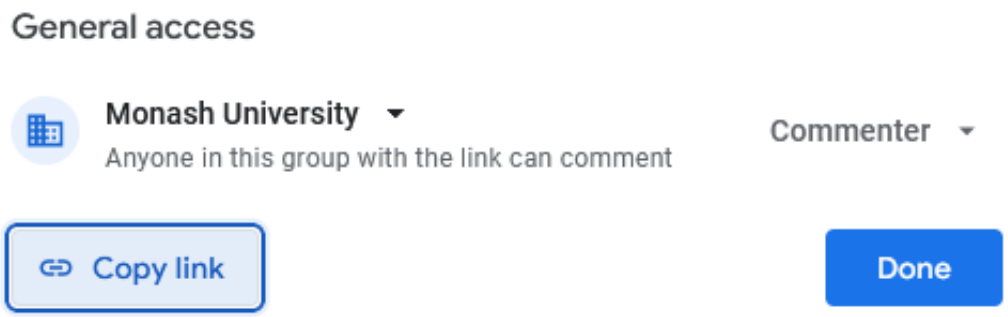
Contribution Logs Template

In a group assessment, individual contributions matter. Each of you will contribute about equivalent number of works depending on what the team agrees. To ensure the accountability of each team member, we need to have written proof that records individual contributions. You'll use it to track your team's progress and give approximate feedback to whoever lagging behind in your team.

Instructions

FIT2099 Contribution Logs Spreadsheet Template:
https://docs.google.com/spreadsheets/d/1-UHjoM1KDm6ZJNBzryJ4vlbG2RR33juYE5zBcXail5M/edit?usp=sharing

1. Go to our Google Spreadsheet above and copy it inside your Google Drive **(with your university email address)**
2. On the top right corner, click "Share" and ensure that "General Access" has the following setting:



3. Click "Copy Link" and paste this link to the README file at the root of your repository.
4. Please ensure that you read through the instruction that is provided in the template.

You don't have to record any programming contributions in this Google spreadsheet because we will check them via git logs (specifically, by using the contributions & graph menu inside Repository). Please be aware that we only check `main` branch, and we may miss contributions that are done in the other branches. Hence, ensure to merge all commits from other branches into the `main` branch before the deadline.

We highly recommend that tasks are small enough to be completed daily. We also provide a column to write down the reflection of the task (i.e., extra notes). You may think of it as journalism, one of the recommended approaches in active learning. We strongly believe that this tool gives you immediate feedback on your learning progress compared to waiting for feedback from us.

Always remember, this is a team effort, so each individual has responsibility for ensuring the balance of contributions' quantity and quality.

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, and 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 the 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

If you notice there's ambiguity in requirements, please don't hesitate to post a question in the forum, or visit available consultations. Regarding edge cases in the scenario, you shouldn't worry much because that won't be our priority. As long as the game is playable with minimum stated requirements, you're good. We always welcome the expansion of new features as long as they don't contradict stated requirements. For ease of mind, please write down your assumptions in the README and notify your interviewer about it.

Design & Diagrams

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 weeks of the semester. Interaction diagrams are covered in Workshops too. 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** (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). We cannot guarantee that your marker will have the same tools available that you used (or the same versions).

If you want a UML diagramming tool, there are a few suggested tools under "Assignment Resources" in Moodle (see Assignments).

As you work on your design, you must store your design diagrams in your Git repository. This will help your marker verify that you have been completing your share of the work on time as agreed.

Severe penalty will be applied for not working on design documents (especially in Assignment 3).

Class diagrams

We expect that you will produce *class diagrams* that cover new or changed classes that implement new features. **Your class diagrams must not 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 in *UML Distilled* (3rd edition).

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 must NOT 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/requirement (**ideally, one diagram should represent the design for one requirement**, not one package). You should show all classes and relationships (e.g., dependency, association, generalisation, and realisation) within a 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

Only applicable for assignment 2 and assignment 3. Not applicable for assignment 1.

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 Applied Session demonstrator (TA).

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

In other units, especially in business process analysis domain, 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 involves 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.

HENCE, you MUST **NOT** draw a system interaction diagram. It means, you don't have to include all written requirements in one interaction diagram. So, please pick at least one interesting and complex feature (e.g., buying an item, talking to someone, consume food, coin stored as credits, and more) and draw an interaction diagram out of it.
1 feature = 1 interaction diagram.

Hint: Good candidates for interaction diagrams often include complex playTurn() or Behaviour operations, which 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.

For example, the Player picks up a coin, and that coin is converted to credit and stored in the wallet, how are those classes that you have designed in the UML class diagram will **interact** with each other?

Dealing with Special Considerations

If you face difficulties completing the assignment (e.g., sick), you may submit special consideration a maximum of a day before the deadline. Please use the Google Form provided in each Assignment instruction for a 5-day extension.

> ⚠ IMPORTANT: Special considerations are **individual**. The rest of the team still need to submit by the set deadline.

So... one of your team members gets sick, what to do?

The rest of the team members must organise the work by splitting the work equally. We recommend a student who received this special consideration work on remaining features and final touch-up on the code & documents during the provided period. We will apply **10% of the available mark per *late* day** to individuals who don't have special considerations and make modifications to the `main` branch after the official deadline.

How do we mark you in such a case?

For the remaining team members, we will use `checkout` feature and mark your work on the original deadline. If we notice the quality is poor or not working properly, we will `checkout` the latest commit that we think is relevant to that assignment. In this situation, if there are changes in the `main` branch beyond the official deadline from anyone who doesn't have special consideration, we will automatically apply late penalties to those students.

If you want to start working on the next assignment, ideally, the team must create a separate branch (e.g., `assignment-2`). It is similar to Bootcamps marking, where we need to check the latest version in `main` the branch and ignore other branches. Make sure your changes don't influence `main` the branch for current assignment marking. So, your team is responsible for ensuring the correctness of code and document versions in that branch.

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 (**Assignment 1**), 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, the iterative process (start from design, test that idea in the code, and update the design) will definitely help you develop a good habit to understand the learning materials better. So, you can achieve desired learning outcomes.

This means that, while you address Assignment 1, please, start getting familiarised with the code provided. You can also begin to implement some basic functionalities to understand the **engine code** and create an informed class diagram. 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 your code until Assignment 2.

Please keep in mind that you MUST use the provided game engine codes. They may not be optimal, but they should make the development of your code easier. So, don't write the game mechanism codes on your own (e.g., positioning logic, action mechanism, etc.). Why? Because we want you to practice using a legacy code/framework since it is an important skill in the industry.

Submission Instructions and Marking Procedures

Submission instructions

Unless a team member has applied for and received special consideration according to the Monash Special Consideration Policy, late submissions will be penalized at 10% per day late (including weekends). Special considerations are **personal**. Therefore, team members without special consideration need to submit their parts by the original deadline.

It is all 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.

Additionally, please, **one team member should submit the following to Moodle** (Assignments) before or slightly after the deadline.

- For Assignment 1, **a PDF file** that contains all your UML diagrams + rationale, and a copy of the contribution logs spreadsheet.
- For Assignments 2 and 3, **a zipped version of your whole repository** (incl. diagrams, rationale, and code)

Your marker will mark the latest version pushed to your repository, not the one in Moodle, but the above copies need to be uploaded to Moodle for the assignments to be marked.

Important: Before being able to upload your files in Moodle, please, complete the corresponding **Feedback Request** form.

Feedback Request forms

In FIT2099, we want to maintain a dialogic feedback culture that involves both peer feedback, and personalised teacher feedback. Therefore, we have introduced a new component to our assignment submission pack: THE FEEDBACK REQUEST SURVEYS

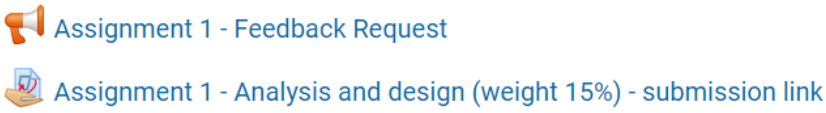
These surveys will let you provide information to us regarding:

- The quality of our materials and support,
- Your expectations on the feedback you want to receive,
- Your experience with previously provided feedback.

Before being able to submit your files, please, complete the corresponding feedback request survey. For example, for Assignment 1 you will see a link in Moodle shown as "Assignment 1- Feedback Request":

After clicking on this link you will see the survey. Responding to the survey will take less than 5 minutes to complete and they are simple and easy to understand. Please, make sure you press the button "Submit your answers" once you finalise the survey:

After that, the Link to upload your files will be available. For example, for Assignment 1, it will look like this:



Importantly, these surveys **will not reward or remove any mark from your assignment**. Instead, they will create an opportunity to flag the main areas of your work you want to receive targeted feedback on from your markers.

So do not ignore them. Take this opportunity and send your responses along with your submissions.

How will you 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. This will be a handover interview that will help you put into practice your communication skills. 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 or that there is a suspected contract breach.

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 class files (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.

Part 1: Introduction & Base code

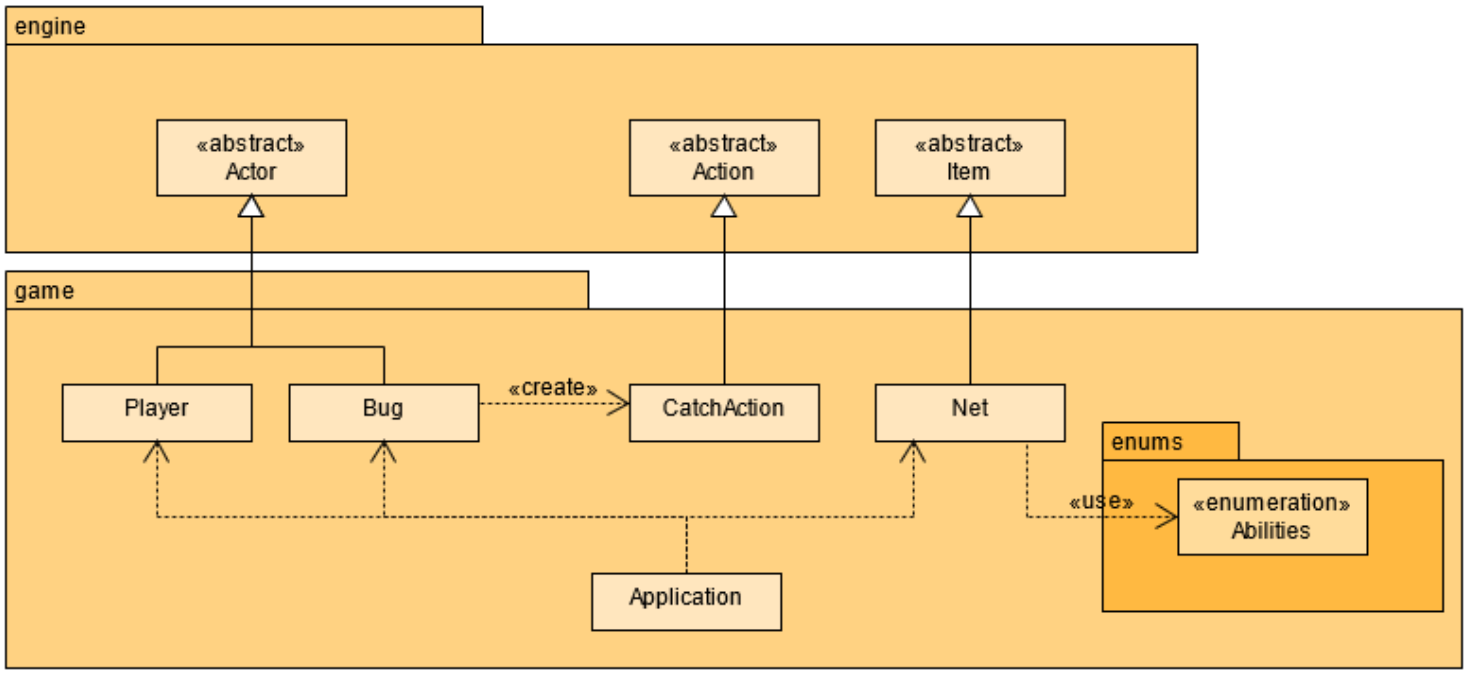
Since we're still working on groups & we have not created Gitlab repositories yet, please refer to Assignment 1 Learning Materials. It is intentional that a shared .zip file doesn't have a `game` package. We don't want you jumping to the code straightaway. Design first, then code, fix it up, repeat

Part 2: Assignment Game Engine

Bug Example (From Part 2) - Design Rationale example

Here is the UML class diagram from the Bug example in Part 2: Assignment Game Engine and design rationale example.

UML Class Diagram



Design Rationale

CatchAction extends Action. All concrete classes that inherit abstract Action class has the necessary methods to provide written text before (`menuDescription`) and after the interaction `execute`.

Bug ---<<create>>---> CatchAction and **Net ---<<use>>---> Abilities.** In this game, the ideal way to interact with the object is by attaching appropriate action to its corresponding object (aligns with the meaning of "object-oriented"). The bug is the object that gives the other Actor (i.e., the Player) an action to be caught. Alternatively, we can create CatchAction inside the Net. However, doing this will require the target Actor (i.e., Bug) inside the Net class to know which actor Player will interact with. By doing so, we will have additional dependency between Net and Actor/Bug, and we also need to check if the Actor is Bug or not. Checking the class type (using if-else statements) is not recommended in the object-oriented as it involves extra dependency. It results in high coupling. Hence, we discard this alternative. Our final design has aligned with the `Reduce Dependency Principle` because it breaks such coupling between Net, Bug, and CatchAction.

The application creates Net instead of creating Net inside the Player's constructor. Both designs are okay and do not break any object-oriented principles. We decided to create Net in the

Application and add it to the Player's inventory because we ensure that the meaning of each class is clear and they can be reused in other scenarios (low coupling & adheres to the DRY principle).

NOTE: You don't have to write such detailed reasoning. As long as you can answer ***WHY*** you do so by elaborating what you've learnt from the learning resources, you'll be fine 😄

More help?!

Please go through our weekly `[Assignment Support] Game  Development Concepts` for extra information. We intentionally put it as part of the weekly lessons because we don't want to overwhelm you here. We'll talk about important jargon and concepts in the first three weeks. Then, we'll talk about code explanation in week 4. Lastly, you'll see a step-by-step approach to developing a mini-game application with our game engine (framework). We also recommend revisiting the workshop/classroom content.

We hope that helps! 😄