# CSSE2002 — Programming in the Large
# Assignment 1 — Semester 1, 2025

School of EECS
The University of Queensland

*Due April $2^{nd}$ 13:00 AEST*

I've never been able to see any boundary between
scientific research and game-playing.
— Donald Knuth

Version 1.0

## 1 Overview

Greetings fellow earthling! We, the governing body of earth (not aliens), require your participation in mandatory earth-protection-focused anti-extraterrestrial-defense training.
Unfortunately we, the governing body of earth (still not aliens), have lost our copy of Atari's 1978's hit cultural phenomenon *Space Invaders*, and require you {*subject_name*} to assist in the creation of a suitable alternative.
Congratulations, {*subject_name*}! Everyone in {*subject_home_town*} would be very proud and impressed with you!

Given our earthling laywers' advice that "Star wars" would upset other earthling laywers, we have decided to name our new implementation $^{Not}$*Space Invaders*.

In your creation process, you will be guided and evaluated. Your task and evaluation is detailed below.

**Common Mistakes** Please carefully read *Appendix A*. It outlines common and critical mistakes which you must avoid to prevent a loss of marks. If at any point you are even slightly unsure, please check as soon as possible with course staff.

**Plagiarism** All work on this assignment is to be your own individual work. Code supplied by course staff (from this semester) is acceptable, but must be clearly acknowledged. Code generated by third-party tools is acceptable but must be clearly acknowledged. See Generative Artificial Intelligence below. You must be familiar with the school policy on plagiarism:

`https://uq.mu/rl553`

**Generative Artificial Intelligence** You are **strongly** discouraged from using generative artificial intelligence (AI) tools to develop your assignment. This is a learning exercise and you will harm your learning if you use AI tools inappropriately. Remember, you will be required to write code by hand in the final exam. If you do use AI tools, you must clearly acknowledge this in your submission. See *Appendix B* for details on how to acknowledge the use of generative AI tools.
Even with acknowledged AI use, you must be able to explain all parts of your submission.

**Interviews** In order to maintain assessment integrity and in accordance with the course profile, you may be asked by the course coordinator via email to attend an interview to evaluate genuine authorship of your assignment. Please refer to the course profile for further details.

## 2 Getting Started

To get started, download the provided code from the following location:

`https://csse2002.uqcloud.net/assessment/A1/provided.zip`

After that, you are **strongly** encouraged to create a diagram mapping out the relationships between all the classes you must implement (details of which are in the javadoc).

### Component #1: Implementation

You are required to implement all of the classes and interfaces specified in the provided Javadoc. Do NOT modify any package, class, or method that is tagged as provided (such as the `game.ui` package or the `GameModel.setRandomSeed` method). Stubs (such as `GameModel.java`) have been given to you to help start you on the right path, but the class implementation must be finished.

`https://csse2002.uqcloud.net/assessment/A1/javadocs/index.html`

The staging below is intended to guide and assist you in implementation, but it remains your responsibility to ensure that you have completed the implementation.

### Component #2: Style

You will be required to write well-styled code throughout the assignment as specified in the Style Guide on Blackboard. Further details in the *Marking Breakdown*.

## 3 Stages

Your assignment has been split into stages that help deconstruct the task into managable sections, and allow you to notice progress before reaching complete implementation. Stages are optional, but recomended. You may complete the assignment in any way you prefer, but the stages are designed to save you time and minimize frustration. If you do not complete all stages you may still score high marks as not all stages are required to pass. For more details, see the *Marking Breakdown*.
This assignment has been split into the following stages:

- Stage 0: A stray Bullet.
  - Implement Bullet class and its parents
  - Implement Logger interface
  - Implement GameModel.getSpaceObjects and GameModel.addObject
  - In GameController.renderGame call ui.render with GameModel objects, and uncomment renderGame in GameController.onTick
- Stage 1: Contact with the Enemy
  - Implement Enemy class and its parents
  - Implement GameModel.updateGame and GameModel.checkCollisions, and uncomment them in GameController.onTick
  - Implement Asteroid
  - Comment/Uncomment in GameController.startGame
- Stage 2: Taking Control
  - Implement Ship class and its parents
  - Implement GameController.handlePlayerInput and anything else it depends on or makes use of.
  - Implement the remaining methods required to instantiate the ship (hint: look in GameModel). You can now control the ship!
  - Comment/Uncomment in GameController.startGame
- Stage 3: Level Up
  - Implement any and all remaining classes, interfaces, gameplay functionality and gameplay systems. For example, levels, spawning, scoring, health, rendering etc.

Maintain and improve code style and documentation throughout!

## 4 Marking

The assignment is marked out of 100. The marks are divided into 2 categories: functionality ($F$), and style ($S$).

| | Weight | Description |
|---|---|---|
| $F$ | 90 | The provided code has been extended to include the specified new components and those components function as expected. |
| $S$ | 10 | Code style conforms to course style guides. |

The overall assignment mark is defined as

$$A_1 = (90 \times F) + (10 \times S)$$

**Functionality** Each class has a number of unit tests associated with it. Your mark for functionality is based on the percentage of unit tests you pass. Assume that you are provided with 10 unit tests for a class, if you pass 8 of these tests, then you earn 80% of the marks for that class. Classes may be weighted differently depending on their complexity. Your mark for the functionality, $F$, is then the weighted average of the marks for each of the $n$ classes,

$$F = \frac{\sum_{i=1}^{n} w_i \cdot \frac{p_i}{t_i}}{\sum_{i=1}^{n} w_i}$$

where $n$ is the number of classes, $w_i$ is the weight of class $i$, $p_i$ is the number of tests that pass on class $i$, and $t_i$ is the total number of tests for class $i$.

**Code Style** The Code Style category is marked starting with a mark of 10. Every occurrence of a style violation in your solution, as detected by *Checkstyle* using the course-provided configuration[1], results in a 1 mark deduction, down to a minimum of 0. For example, if your code has 2 checkstyle violations, then your mark for code quality is 8. Note that multiple style violations of the same type will each result in a 1 mark deduction.

$$S = max(0, 10 - \text{Number of style violations})$$

Note: There is a plug-in available for *IntelliJ* which will highlight style violations in your code. Instructions for installing this plug-in are available in the Java Programming Style Guide on Blackboard (Learning Resources → Guides). If you correctly use the plug-in and follow the style requirements, it should be relatively straightforward to get high marks for this section.

### Electronic Marking

Marking will be carried out automatically in a Linux environment. The environment will not be running Windows, and neither IntelliJ nor Eclipse (or any other IDE) will be involved. OpenJDK 21 with the JUnit 4 library will be used to compile and execute your code and tests. When uploading your assignment to Gradescope, ensure that Gradescope says that your submission was compiled successfully.

Your code must compile.

If your submission does not compile, **you will receive zero marks**.

[1]The latest version of the course *Checkstyle* configuration can be found at `http://csse2002.uqcloud.net/checkstyle.xml`. See the Style Guide for instructions.

## 5 Submission

Submission is via Gradescope. Submit your code to Gradescope *early and often.* Gradescope will give you some feedback on your code, but it is not a substitute for testing your code yourself.

You must submit your code *before* the deadline. Anything that is submitted after the deadline will **not** be marked (1 nanosecond late is still late). See *Assessment Policy*.
You may submit your assignment to Gradescope as many times as you wish before the due date. Your last submission made before the due date will be marked.
You must submit ALL files you create or modify, and NONE of the files you do not.

### Code Submission

Your code submission must include at least the following directories:

**Implementation Code Files:**

```
src/game/core/*
src/game/exceptions/*
src/game/utility/*
ai/README.txt
```

**Do not** include the provided code outside of these packages! If you create additional packages, include them in the submission. **Only submit code you have modified or created**.

Ensure that your classes and interfaces correctly declare the package they are within. For example, `GameModel.java` should declare `package game.model;`.

**Provided tests** A small number of the unit tests used for assessing Functionality (F) are provided in Gradescope, which can be used to test your submission against.

The purpose of this is to provide you with an opportunity to receive feedback on whether the basic functionality of your classes and tests is correct or not. Passing all the provided unit tests does *not* guarantee that you will pass all the tests used for functionality marking.

## 6 Assessment Policy

**Late Submission** Any submission made after the grace period (of one hour) will not be marked. Your last submission before the deadline will be marked.

Do not wait until the last minute to submit the final version of your assignment. A submission that starts before the end of the grace period but finishes after will not be marked.

**Extensions** If an unavoidable disruption occurs (e.g. illness, family crisis, etc.) you should consider applying for an extension. Please refer to the following page for further information:

`http://uq.mu/rl551`

All requests for extensions must be made via my.UQ. Do not email your course coordinator or demonstrators to request an extension.

**Remarking** If an *administrative error* has been made in the marking of your assignment (e.g. marks were incorrectly added up), please contact the course coordinator (csse2002@uq.edu.au) to request this be fixed.

For all other cases, please refer to the following page for further information:

`http://uq.mu/rl552`

## A Critical Mistakes

### THINGS YOU MUST AVOID

Code may run fine locally on your own computer in IntelliJ, but it is required that it also builds and runs correctly when it is marked with the electronic marking tool in Gradescope. Your solution needs to conform to the specification for this to occur.

- Files must be in the correct directories *(exactly)* as specified by the Javadoc. If files are in incorrect directories *(even slightly wrong)*, you will lose marks for functionality in these files because the implementation does not conform to the specification.
- Files must have the exact correct package declaration at the top of the file. If files have incorrect package declarations *(even slightly wrong)*, you will lose marks for functionality in these files because the implementation does not conform to the specification.
- You must implement the public members exactly as described in the supplied documentation (*no extra public members or classes*). Creating public data members in a class when it is not specified will result in loss of marks, because the implementation does not conform to the specification.
  - You are encouraged to create private members and protected methods as you see fit to implement the required functionality or improve the design of your solution.
- Do not use any version of Java newer than 21 when writing your solution. If you accidentally use Java features which are only present in a version newer than 21, then your submission may fail to compile.

## B Generative Artificial Intelligence

While the use of generative AI for this assignment is discouraged, if you do wish to use it, ensure that it is declared properly.

For this, you must create a new folder, called "`ai`", within this folder, create a file called "`README.txt`". This file must explain and document how you have used AI tools. For example, if you have used ChatGPT, you must state this and provide a log of questions and answers received using the tool. The "`README.txt`" file must provide details on where the log of questions and answers are within your "`ai`" folder.

If you plan to use continuous AI tools such as Copilot, you must ensure that the tool is logging its suggestions so that the log can be uploaded. For example, in IntelliJ, you should enable the log by following this guide: `https://docs.github.com/en/copilot/troubleshooting-github-copilot/viewing-logs-for-github-copilot-in-your-environment` and submit the resulting log file.