# SENG201 – Software Engineering I
# Project Specification
# Monster Fighter

For project admin queries:
Matthias Galster — Miguel Morales — Morgan English
{matthias.galster, miguel.morales, morgan.english}@canterbury.ac.nz

For technical support, project help, hints and questions:
Danita Sun — Callum McLoughlin —
Ed Langlands — Sam Shankland
danita.sun@canterbury.ac.nz, {cwm62, ell30, sjs227}@uclive.ac.nz

March 29, 2022

## 1 Introduction

### 1.1 Administration

This project is a part of the **SENG201** assessment process. It is worth **30%** of the final grade. This project will be done in pairs (i.e., teams of two students). You must find your own partner and register the team on LEARN. Submissions from individuals will not be accepted. Pairs are not allowed to collaborate with other pairs, and you may not use material from other sources without appropriate attribution.

Plagiarism detection systems will be used on your report and over your code, so do not copy the work of others. Plagiarised projects will be referred to the University proctor for disciplinary actions. Your deliverables must be submitted on LEARN. Students will be asked to demo their project during lab and lecture times.

Actual dates and times are detailed in Section 6.3. The drop dead policy and other penalties are detailed in Section 6.4.

You can find all the project-related information and resources in the "Project" section on LEARN.

## 1.2 Outline

This project will give you an idea of how a software engineer would go about creating a complete application (which features a graphical user interface) from scratch. **In this project you will build a game in which you buy and maintain a party of monsters. You will be able to buy monsters, equipment and food to upgrade their stats; fight with them to earn money; and sleep till the next day to heal and refresh the shop inventory. You will keep fighting and shopping until all days are complete with the goal to earn the highest score you can.**

The idea of the game is slightly open to allow some room for creativity, but please ensure you implement the main project requirements as that is what will be graded. We encourage you to think of the assignment as a *project* rather than a *programming assignment*. Programming assignments often have a clearly defined specification that you can follow step-by-step. However, in practice, software engineering projects require a lot of thinking from the engineers to find out what exactly to build, why, for whom, and how. Therefore, specifications and requirements are often vague and need to be clarified during development.

## 1.3 Project Management

Adequate planning will help you minimize risks which can negatively impact your project (e.g., falling behind, poor testing). Defining time estimates, clear goals and realistic milestones will give you a much higher chance to succeed. To help you with this, you must record the following data weekly:

- Hours you spent working on the project during the week.
- Activities you carried out during the week.
- If you achieved the goals planned for the week or not.
- The risks you identified or faced during the week.
- How satisfied you are with your and your partner's contribution to the project during the week.

A link to record these data is available on LEARN.

You will earn marks by entering the data on time. Similarly, you will lose marks if you fail to regularly record the data, see Section 6.4 for more details.

## 1.4 Help

This project can get confusing and frustrating at times when your code does not work. This will likely be the largest program you have written thus far, so it is **very important** to break larger tasks into small achievable parts, and then implement small parts at a time, **testing as you go**.

Having a nice tight modular application will help with debugging, so having appropriate classes is a must. If you are having problems, try not to get too

much help from your classmates, and instead ask for help from your tutors. You can email them or ask them questions in labs. Always save your work and have backups, do not assume that the CSSE department will be able to recover any lost data.

# 2 Requirements

This section describes what your game must do and is written as a set of requirements. At the beginning of your project and to get started, try thinking of each requirement as a separate ticket that needs to be closed **before** others are started, it will help you have code which works and can be built upon, instead of a lot of broken spaghetti code.

**Hint:** Functionality can be placed in its own package or class. Modularisation is the key, especially when you begin GUI programming.

## 2.1 Setting Up the Game

When your game first starts it should ask the player to:

1. Choose a player name. The length must be between 3 and 15 characters and must not include numbers or special characters.
2. Select how many days they would like the game to last (between 5 and 15 days).
3. Choose a starting monster for your team:
   (a) Each monster should have different characteristics. Including max health, damage, heal amount, current health. Get creative!
   (b) The player shall have 3 to 5 monsters to choose from.
   (c) The player shall be able to name the monster (or have a default name)
4. Choose a difficulty
   (a) There must be at least 2 options.
   (b) This should scale how difficult the game is in a noticeable way. (For example how much gold the player starts with and/or gets for winning battles, or how hard the opponents will be).
   (c) You may want the harder difficulties to give a higher score for the same action.
5. Start your adventure.

## 2.2 Playing the Main Game

Once the player has finished their setup and selected their starting monster, the main game can begin. The player starts out on day one with a default amount of gold, ready to buy new monsters or upgrade them with food and other items. The amount of gold is up to you (and may be scaled with difficulty). The following options will be displayed to the player:

1. View the amount of gold you have, the current day, and the number of days remaining.
2. View the properties of your team. This shall include:
   (a) The name of each monster (this may be a generic name e.g. Infernal Rhino, or a name given by the player).
   (b) The properties of each monster (importantly attack, and current health though all should be viewable).
   (c) It should be clear to the player what order their monsters are in (if this matters for your battling mechanic).
3. View the player inventory, and the current items they have. Each item shall:
   (a) Show the effects of the item.
   (b) Allow for the player to use the item on a monster.
4. View the possible battles
   (a) The player should be able to see a small number (3-5) of optional battles they can take on.
      i. The battles should be generated somewhat randomly, but largely influenced by the current day, and optionally the difficulty setting
      ii. The gold and points gained for winning a battle may scale with the difficulty (e.g. less gold on hard, but more points given)
   (b) The player should be able to choose a battle to fight.
      i. A player can battle each battle once.
      ii. If all of a player's monsters have fainted they can no longer battle.
      iii. If all of a player's monsters faint during a battle they lose the battle and do not receive any gold or points
      iv. If a player wins a battle (by fainting all opposing monsters) they are rewarded with gold and points
5. Battling
   (a) Within a battle monsters should fight in some meaningful way for example:

i. Each side is ordered and the monsters fight in order, where each fight is made up of a series of moves, where the front monsters deal their attack damage to the opposing monster's health. With a monster being removed from the fight when their health is reduced to zero, in which case the next monster is now fighting.

(b) Once a battle has concluded the outcome should be clear, and the updated status of the team should be shown to the player.

(c) You may optionally show each step in a battle, with the current leading monsters and their attack and health.

6. Visit the shop and:

   (a) While at the shop, the player must be able to see their current gold.

   (b) Monsters and items can be sold back to the shop.

   (c) View monsters that are for sale including their price and their properties

      i. There should be 3 to 5 monsters to choose from

      ii. Monsters can only be bought if there is space for them on the team. A team shall have a maximum of 4 monsters

      iii. The type of monsters sold may depend on other factors

         A. E.g. a rarity system could be used, with rarer monsters becoming more common (or being unlocked) in later days

   (d) View items that are for sale including their price and properties

      i. There should be at least three items in the shop

      ii. When an item is bought, it should go to the player's inventory

      iii. Items should enhance monster stats

7. Go to sleep (move to next day)

   (a) All items in the shop are updated (possibly randomly)

   (b) All battles are updated (following 4)

   (c) Each monster heals for their heal amount not exceeding their max health

There will also be some random events you will need to implement. These will only happen overnight. The chances of these events occurring may depend on the difficulty setting. The player should be alerted when any of these random events occur:

1. A monster levels up overnight

   (a) There should be a small chance a monster levels up overnight (you may wish to tie this into other mechanics, such as increasing with the number of battles won, or whether or not the monster fainted that day)

2. A monster leaves overnight
    (a) The chance should increase if the monster fainted the previous day(s)
    (b) The chance should be quite low
3. A random new monster joins overnight
    (a) The chance should increase depending on how many free slots the player has in their team
    (b) The chance should be quite low

## 2.3 Finishing the Game

The game ends when one of the following occurs:

1. All days have passed
2. The player has no monsters and not enough gold to buy another

Upon ending a screen will display the player's name, the selected game duration in days, the amount of gold gained, and the amount of points gained.

## 2.4 Extra Credit Tasks

If you have finished early, and have a good looking application, then you will be in for a very high mark. If you want some more things to do, then please discuss it with the tutors in the lab, but you are free to add any features you wish, just be careful to not break anything. Here are some ideas, and you do **NOT** need to implement them all to get full marks:

- You may want to allow the player to enter a 'seed' during setup that determines how random events of the game will play out
- You may want to show the battles with sprites or animations
- You may want to allow the player to save the current state of the game, and be able to load it up at a later time
- You may want to add a story line to your game, including consistent characters, and a plot which gets told through pop ups or dialogue
- You may want to add rarity levels for monsters which are properly tied in via some mechanic, such as increasing their chance to appear/unlocking them in the shop after some milestone (e.g. number of days or number of battles won)
- You may want to add special characteristics to monsters to make them more unique

  - For example, a monster may not heal overnight, but start with higher base stats, then disappear when it faints, or a monster may get one free hit each battle. These effects are very open ended, but you may want to think about a good framework for your battles to have these work well without creating spaghetti code

- You may want to add some artwork
- You may want to include some music or sound effects

# 3 Design and Architecture

This section provides some **ideas** on how your program should be structured. The following are some class candidates, and should be represented in your program. Important things to think about here are interactions between classes, what classes should be instantiated, how you could use interfaces and/or inheritance, and how you could apply the design patterns you've learnt in SENG201.

## 3.1 Monster

There shall be different types of monsters. Six shall be enough. Each monster has at least a name, max health, damage, heal amount and current health.

## 3.2 Item

There shall be different types of items (e.g., different potions and foods). Four shall be enough. Each item should increase one or more of a monster's properties when consumed.

## 3.3 Purchasable

Both monsters and items can be bought from a shop. These both require a purchase price, a sell-back price, and a description.

## 3.4 Battle

Battles shall be generated based on some game factor (e.g., current day or difficulty) with some randomness. A battle may be with another 'player', in this case this other 'player' must have a name. A battle consists of a number of monsters that fight against the player's current team.

## 3.5 Random Event

A random event represents a circumstance that can happen overnight. It should include behaviour to handle the event.

### 3.6 Game Environment

The game environment contains your game, and will implement functions to provide the options mentioned above, and will call methods of the above classes to make those options happen. The game environment keeps track of the game and should handle requests from the user interface and report back updates.

Try to keep your code modular and avoid putting user interface (UI) code in your game environment class. Keeping the game logic separate from the UI code will ensure that you can create both a command line interface and a GUI interface without having to alter your game environment.

## 4 Assignment Tasks

### 4.1 Sketching UML

**Before** you start writing code, sketch out a UML use cases diagram detailing the program's users (actors) and their interactions with the system (use cases). This diagram will help you to identify the main functionalities of the program and to visualize its scope.

In addition to this diagram, create a UML class diagram of how you think your program will look like. It will help you get an idea of what classes there are, what class attributes are required, and will get you thinking of what classes communicate with other classes (call methods of another class).

### 4.2 Implementing a Command Line Application

Begin implementing classes, starting with monster, player, store, items, and the game environment. Make a command line UI class that drives a simple command line application that works in a simple runtime loop which:

1. Prints out a list of options the player may choose, with numbers next to the options.
2. Prompt the player to enter a number to complete an option.
3. Read the number, parse it and select the correct option.
4. Call the method relating to the option and if necessary:
   (a) Print out any information for that option, such as select a battle.
   (b) Read in the number for the information offered.
   (c) Parse the number and complete the action.
5. Go back to step 1.

This will enable you to slowly build up features, and we recommend to only implement one feature at a time. Make sure you test your feature before moving

onto implementing more features. Once you are feature complete and have a working game, you may move onto implementing a graphical application.

The command line application will only be assessed if there is no graphical application, or if there are fatal bugs in the graphical application. In this case, a penalty will be applied (see Section 6.4). **Hint:** For a very basic solution to read input from the command line you may want to explore class Scanner in the Java API.

## 4.3 Implementing a Graphical Application

You will be implementing a graphical application for your game using **Swing**, which will be explained in labs. For the purposes of this assignment, we do not recommend writing the Swing code by hand, and instead using the interface builder offered by the Eclipse IDE. This lets you build graphical interfaces in Swing by dragging and dropping components onto a canvas onscreen, and it will automatically generate the code to create the graphical application you built.

Please note, you are required to ensure that any automatically generated code complies with the rest of your code style, so you will need to change variable/method names and code layout.

Once you have built your interface, the next task is to wire up the graphical components to the methods your command line application supplies, and to update the onscreen text fields with the new values of your class attributes/member variables. Most of these functions are triggered on `onClick()` methods from buttons. Start small, and complete Section 2.1 "Setting up the Game" first to get used to GUI programming. You might need to slightly adjust your methods to achieve this. Then move onto the main game.

Note that this is the largest task to complete and many students under estimate how much time it will take. Try to be at this stage one week after the term break if possible.

## 4.4 Writing Javadoc

Throughout your application, you need to be documenting what you implement. Each attribute of a class should have Javadoc explaining what its purpose is. Each method needs to explain what it does, what variables it takes as parameters, and what types those variables are. You should be building your Javadoc regularly, as it integrates into the IDE very nicely, and will aid you in writing good code.

## 4.5 Writing Unit Tests

You should design JUnit tests for your smaller, basic classes, such as Monster, Item, Store, and their descendants if you think necessary. Try and design useful tests, not just ones that mindlessly verify that getters and setters are working as intended.

### 4.6 Report

Write a short two page report describing your work. Include on the first page:

- Student names and ID numbers.
- The structure of your application and any design decisions you had to make. We are particularity interested in communication between classes and how interfaces and/or inheritance were used. You might want to reference your UML class diagram.
- An explanation of unit test coverage, and why you managed to get a high/low percentage coverage.

Include on the second page:

- Your thoughts and feedback on the project.
- A brief retrospective of what went well, what did not go well, and what improvements you could make for your next project.
- The effort spent (in hours) in the project per student.
- A statement of agreed % contribution from both partners.

### 4.7 A note on effort distribution

The "typical" or "common" distribution of the overall effort spent on these activities is: around 5% creating the UML diagrams; 20% developing the command line application; development of the graphical application is the most time consuming task taking around 50% of your time; documenting your code would take 5%; creating the unit tests around 15% and writing the report would take the last 5%.

However, note that these numbers vary between students and these percentages are not supposed to be the exact amount of effort invested in each task. They are only a rough guideline (e.g. we would not expect you to spend 50% of your time on creating UML diagrams or writing the report; on the other hand, we would expect that implementing the graphical user interface takes quite a substantial portion of the effort).

## 5 Deliverables

### 5.1 Submission

Please create a ZIP archive with the following:

- Your source code (including unit tests). We want your exported project as well so that we can easily import it into Eclipse.
- Javadoc (already compiled and ready to view).

- UML use case and class diagrams as a PDF or PNG (do not submit Umbrello or Dia files; these will not be marked).
- Your report as a PDF file (do not submit MS Word or LibreOffice documents; these will not be marked).
- A README.txt file describing how to build your source code, import your project into Eclipse and run your program.
- A packaged version of your program as a JAR. We must be able to run your program **in one of the lab machines** along the lines of: `java -jar usercode1_usercode2_MonsterFighter.jar`.

Submit your ZIP archive to LEARN before the due date mentioned in Section 6.3. **Only one member** of the team is required to submit the ZIP archive.

## 5.2 Demos

During the last week of term, you will be asked to demo your project during lab and lecture time. Each team member must be prepared to talk about any aspect of your application, we will be asking questions about any and all functionality. There will be a form on LEARN in which you can book a time slot, ensure you are both available, as you must attend the demo as a pair.

**Only one member** of the team needs to book a slot for the team, but both members are expected to attend the demo at that time; do not double book slots.

# 6 Marking scheme

## 6.1 Overall Assignment [100 marks]

### 6.1.1 Functional suitability and Usability [45 marks]

We will be testing the extent to which your code meets the requirements using the graphical interface. This includes running the program and executing its main functionalities.

If your graphical application is broken or faulty, partial marks will be awarded for your command line application.

### 6.1.2 Code quality and Design [20 marks]

We will be examining the code quality and design of your program. This aspect include: your naming conventions, layout and architecture, and use of object oriented features, among others.

Quality of your comments is also very important. You may wish to run `checkstyle` over your code before you hand it in.

#### 6.1.3 Documentation [10 marks]

We will be looking at your use of Javadoc, and how well it describes the attribute or method you are commenting on, and how well it covers your codebase.

#### 6.1.4 Testability [15 marks]

We will run your unit tests to see how well they cover your code, and we will examine the quality of those tests. Try to make your tests do something other than verifying that getters and setters work.

#### 6.1.5 Report and UML diagrams [10 marks]

Your report and UML diagrams will be marked based on how well written the report is and the information the diagrams convey about your program. The report must include what is specified in section 4.6.

Employers place a lot of value on written communication skills, and your ability to reflect on a project, especially in agile programming environments.

### 6.2 Project Demos

For your project demo, you will have to log in to a lab machine, download your project submission from LEARN and import it into Eclipse. From there, you and your partner will be showing the examiners how your program works and you may be asked to:

- Construct a new game, and choose a monster.
- View your inventory.
- View the properties of your team.
- Choose a battle and fight.
- Visit the shop and view, buy and/or sell items.
- Show that random events occur.
- Show that the game can be completed.
- Show that the game runs without errors, obvious bugs or crashes.
- Fulfils any or all of the requirements set.
- You may be asked to explain how your graphical interface is written, and point to specific code.
- You may be asked about how interfaces and/or inheritance work in your program, again pointing to specific code.
- Anything else that the examiner wishes to ask about.

If you do not turn up to demo in your time slot, you will be penalised (see Section 6.4). The project demo instructions may change depending on multiple factors (e.g., COVID-19 restrictions), in case of any changes, this will be informed through LEARN.

## 6.3 Important Dates

- **Project launch:** March 29, 2022.
- **Weekly progress deadlines:**
  - **Week 6** (29 March - 4 April, 2022) progress must be recorded by April 6, 2022 at 11:59pm.
  - **Week 7** (5-11 April, 2022) progress must be recorded by April 13, 2022 at 11:59pm.
  - **Week Break-1** (12-18 April, 2022) progress (if any) should be recorded by April 20, 2022 at 11:59pm.
  - **Week Break-2** (19-25 April, 2022) progress (if any) should be recorded by April 27, 2022 at 11:59pm.
  - **Week Break-3** (26 April - 2 May, 2022) progress (if any) should be recorded by May 4, 2022 at 11:59pm.
  - **Week 8** (3-9 May, 2022) progress must be recorded by May 11, 2022 at 11:59pm.
  - **Week 9** (10-16 May, 2022) progress must be recorded by May 18, 2022 at 11:59pm.
  - **Week 10** (17-23 May, 2022) progress must be recorded by May 25, 2022 at 11:59pm.
- **Last day for registering teams:** April 8, 2022 at 5pm. After this time, the teaching team will randomly allocate those students without a pair. Allocations will be final.
- **Last day for booking a time slot for the demo:** May 26, 2022 at 5pm. After this time, the teaching team will allocate time slots to those teams without a booked slot. Allocations will be final.
- **Project submission due date:** May 23, 2022 at 5pm.
- **Lab demos:** during lab and lecture time of 30 May - 3 June, 2022.

## 6.4 Penalties

- **-6 marks** for failure to register your team by the given deadline.

- **-4 marks** for failure to record your weekly progress by the weekly deadline, see Section 6.3 for details. This penalty applies as soon as the project starts and regardless of whether or not you have a team. The penalty will be applied only to the student in the pair who failed to record their weekly progress.

  The penalty applies as many times as you fail to record progress. For example, if you missed it twice, a penalty of -8 marks is applied (2 x (-4) = -8).

  It is not allowed to record the weekly progress in one go at the end of semester.

  Students who record their weekly progress in a timely manner for all weeks (excluding the break weeks) will be granted a one-time bonus of +5 marks. This will only apply to the student in the pair who recorded their weekly progress not later than two days after the reported week ended.

  You are not expected to work on the project during the term break. However, if you decide to do this, it would be beneficial for you to keep recording your weekly progress.

- **-6 marks** for failing to book a time slot for the demo by the given deadline, see Section 6.3 for details. It applies to both members of the team. This penalty also applies if both members of the team book two time slots.

- **-15 marks** for submitting after the due date PLUS **-1 mark** per each hour of delay. It applies to both members of the team and it is capped up to -100 marks, i.e., the total marks of the assignment.

  For example, if you submit 16 hours* after the due date, a penalty of -31 marks will be applied [(-15) + (-16) = -31].

  * For the purposes of this calculation, the ceiling function is used.

- **-30 marks** for not providing a GUI for your application.

- **-30 marks** for failing to show up to the demo. It applies only to the student who missed the demo.

- **-100 marks** if plagiarism is detected.

# 7 FAQ

**I cannot see my group number, can you please let me know either what it is or how to figure it out?**

Your group number is the three-digit number in the group description.

**I can't find the Participants section, could I be pointed in the right direction?**

Open the normal SENG201 LEARN page, then click the hamburger icon (3 horizontal bars icon) in the top left and it should be under that. If you still can't find it, use CTRL+F and search "Participants"

**I submitted my project three seconds past the deadline, is it counting as a late submission?**

Yes, please refer to the subsection 6.4.

**We were just submitting the project while we were disconnected from the WiFi. This made our submission 15 seconds late. Is there any chance we can not be marked down for this?**

No, please refer to the subsection 6.4.

**I forgot to pick a time slot for my demo, could you reopen the demonstrating slot chooser?**

No, please refer to the subsection 6.4. Your demo slot will be allocated by the teaching team.

**Our ZIP file containing our report, Javadoc, UML class diagram and a .JAR file was submitted, however, we forgot to add the source code. Is our submission incomplete?**

Yes, the submission is incomplete. Not submitting your source code will void your submission (i.e., you will only be able to get the marks associated to the Report, see subsection 6.1).

**We are currently having difficulties with LEARN. Is there any chance we could ask for a small extension of around 15 minutes to fix this issue?**

No, please refer to subsection 6.4.

**We realized that we forgot to add the project report, so we had to upload our project again. This resulted in our submission missing the deadline by a few seconds. Is it possible to get the 15-mark penalty removed?**

No, please refer to subsection 6.4.

**We are thinking of deleting the CLI code from the final deliverable as the GUI runs fine. It's [the CLI] not being used at all anymore. Is this fine?**

Yes, it is OK to remove the CLI code if your GUI is working well. However, only the files you submit will be marked. In case your GUI fails, you won't be able to use your CLI project.

**My group partner has booked a time slot, do I need to book in as well?**

No, only one member of the team is required to book a time slot.

**We're getting pretty close to finishing our project, and we'd like to maybe put in some background music to go with our GUI application. What would we have to do with respect to copyright if we wanted to include this?**

The safest path to follow is not using copyrighted material. We suggest finding music under royalty-free or creative commons licenses.

**Is it necessary for the project to be able to be imported into Eclipse so that the markers can build it, or can we specify in our README that it is an IntelliJ project?**

Yes, it is required to be able to import your project into Eclipse. The README may include instructions for IntelliJ but must include the Eclipse ones too.

**How many marks is it possible to achieve for just submitting a command line application of the game?**

You would be losing marks (see subsection 6.4), but the overall marks also depend on the rest of the application, how much functionality you implemented, the quality of your code, testing practices, etc.

**Does our command-line app need to be included in the final class diagram?**

No, the CLI app (and its related documentation, if any) will not be marked unless the GUI does not work.

**If your getters and setters don't contain any logic (i.e., they are trivial) then there is no need to test them thoroughly.**

You can test this type of getters/setters (those without any logic) in conjunction with more complex methods.

**About the class diagram, do we need to include the launch screen and close screen methods in the class diagram.**

There is no need to include those methods (trivial getters, setters, launch, close) in the class diagram.

**I am wondering whether or not my SENG201 project partner and I must submit a report each, or one report between us.**

You must submit one report per team.