De La Salle University
College of Computer Studies
Department of Software Technology

Fatal Fantasy: Tactics

CCPROG3 MCO1 Specifications (AY 2024-2025, Term 3)

1 Project Introduction & Overview 2
2 MCO1: Core Game Logic + Text-Based Simulation 2
2.1 Program Functionalities 2
2.1.1 Character Management 2
2.1.2 Character Creation 3
2.1.3 Text-Based Core Game 3
2.1.4 Interface 4
2.2 Bonus Points 4
2.3 Implementation Requirements 4
3 Documentation 5
3.1 Internal Code Documentation 5
3.2 Test Plan 5
4 Deliverables 5
4.1 Submission & Demo 5
4.2 Deliverables Checklist 5
5 Collaboration and Academic Honesty 6
6 Grading 6
Acknowledgments / Disclosure 6
Appendix A: Declaration of Original Work Template 7
Appendix B: Test Plan Format Sample 8
Appendix C: Export Test Plan Guide 9
Appendix D: Class Options and Abilities 10

Release Date: May 28, 2025

Due Date: June 28, 2025 (Sa) 2100

Prepared by: Kristine Kalaw

1 Project Introduction & Overview

You must create a Java program based on the project specifications detailed below. This project is a culminating program activity that showcases the programming concepts learned in class. Through the project, students should be able to demonstrate the learning outcomes indicated in the syllabus. This project is split into two phases: MCO1 and MCO2.

This project is designed to be completed in pairs (i.e., with a maximum of two members per group). In exceptional cases, the instructor may permit a student to work individually; however, this requires prior approval, as collaboration is an essential part of the learning experience. Under no circumstances will groups of more than two members be allowed.

Whether working individually or as a pair, group composition must remain consistent throughout the entire project. Partner changes between Phase 1 and Phase 2 are not permitted. If a pair separates for any reason, both students must continue the project individually; they may not join another group or form a new pairing.

The project is designed to progressively showcase Object-Oriented Programming Principles. This will be done in two main phases: MCO1 will focus on application design through a UML class diagram and a text-based game simulation via the Command Line Interface (CLI). MCO2 will then culminate in a full GUI-based application with extended mechanisms built upon the foundation of MCO1.

Fatal Fantasy: Tactics is a turn-based fighting game where two players build custom characters by selecting from a variety of character options—resulting in unique characters with distinct playstyles. Players engage in tactical combat by strategically managing their Energy Points (EP) to execute actions and deplete their opponent's Health Points (HP).

Key features of the game include:

- **Character Management** – Players can view, add, edit, and delete their customized characters before entering battle.
- **Character Creation** – Players create unique characters by selecting from various character options.
- **Turn-Based Combat Mechanics** – Battles are driven by strategic decisions, with players managing Energy Points (EP) to execute actions and reduce their opponent's Health Points (HP).
- **Combat Log System** – A detailed combat log tracks all game events, providing transparency and insight into each move made during the match.

2 MCO1: Core Game Logic + Text-Based Simulation

2.1 Program Functionalities

You are tasked with creating a text-based game that simulates a match between two players' customized characters.

2.1.1 Character Management

This feature must allow a mechanism for players to manage their own characters:

- **View character/s**
  - Players should be able to view their own characters which includes both a list view and a detailed view.
  - The list view displays a summary of all created characters
  - The detailed view displays all the information about the selected character.

- **Create character**
  - Specifications for this feature are detailed in Section 2.1.2.

- **Edit character**
    - Once a character has been created, the only properties that can be modified are their selected class abilities.

- **Delete character**
    - Players can delete characters as needed.
    - Players can have a maximum of six (6) created characters at any given time.

2.1.2 Character Creation

This feature of the game must observe the following flow:

1. **Class Selection**
    - Determines the character's ability pool, with each class providing a unique set of abilities that define distinct playstyles and strategies. Refer to Appendix D for the list of classes and their corresponding abilities.

2. **Class Ability Selection**
    - Players must choose three (3) class abilities from their class's ability list.

3. **Name the character**
    - This is the name of the character.
    - To make the differentiation of characters easier, names must be unique.

A base character has a maximum of 100 HP (indicates how much damage they can sustain before they are knocked out), a maximum of 50 EP (indicates how much energy they have for executing moves), and a maximum of 3 class ability slots. Note that in addition to class abilities, all characters also have these moves:

- **Defend** (costs 5 EP): The character takes on a defensive stance and takes only half damage
- **Recharge** (no EP cost): The character does nothing during the round but regains 5 EP

2.1.3 Text-Based Core Game

This feature of the game must observe the following flow:

1. **Player 1 character selection**
    - The program prompts who Player 1 is and which of their characters will enter the battle.

2. **Player 2 character selection**
    - The program prompts who Player 2 is and which of their characters will enter the battle.

3. **Fight!**
    - Characters begin each fight at max HP and EP
    - Each round observes the following flow:
        1. At the start of each round, both characters regenerate +5 EP (capped at max)
        2. Display the round number, current HP, and current EP of both characters
        3. Display Player 1's available list of moves, including EP cost
        4. Prompt Player 1 to choose a move, re-prompting them if the choice is invalid (e.g., invalid choice, insufficient EP)
        5. Repeat steps 3-4 for Player 2
        6. Execute moves
        7. Display the round's outcome (e.g., damage dealt, health changes, EP spent)
        8. Check if the battle has ended (the battle ends when a character or both characters has dropped to 0 HP after their moves have been executed)
            - If the battle has concluded, declare the results, and then prompt for a rematch or go back to the main menu
            - Otherwise, the game continues to the next round

2.1.4 Interface

This phase of the project is text-based and should be run from the terminal or command-line. Students have the flexibility to choose the type of text-based user interface they wish to implement, provided it is intuitive and adheres to the specifications outlined in this document. Creativity in presenting the game is encouraged, as long as the specifications are followed.

There must be error checking for all player inputs. If the input comes from a list of options, then only the options should be allowed. If the input requires a string, then the input string must be valid. If the user enters an invalid input, the program should not crash. The program must display an error message and continue to prompt the user until an appropriate input is given.

2.2 Bonus Points

No bonus points will be awarded for MCO1. Bonus points will only be awarded for MCO2. To encourage the usage of version control, please note that a small portion of the bonus points for MCO2 will be the usage of version control. Please consider using version control as early as MCO1 to help with collaborating within the group.

2.3 Implementation Requirements

The project must abide by the following implementation guidelines:

- The program should be **fully implemented in the Java programming language**.
- Observe the appropriate use of programming constructs (e.g., conditional statements, loops, data structures, classes).
  - **Do not use brute force.**
  - Codes must be modular (i.e., split into several logical methods/classes). Codes that are not modularized properly will not be accepted, even if the program works properly.
  - Exhibit proper object-based concepts, like encapsulation and information-hiding
- You can use these in the future when you have much more experience and wisdom in programming. While you are the **in PROG series you are NOT ALLOWED to do the following:**
  - ✘ to declare and use global variables (i.e., variables declared outside any classes)
  - ✘ to use `exit`, `goto` (i.e., to jump from code segments to code segments), `break` (except in `switch` statements), or `continue` statements
  - ✘ to use `return` statements to prematurely terminate a loop, function, or program
  - ✘ to use `return` statements in `void` functions
  - ✘ to call the `main()` function to repeat the process instead of using loops
- You may only use what is available in the Java API
- You may use topics outside the scope of CCPROG3, but this will be **self-study**.
- Adhere to coding conventions and **must include proper documentation** as indicated in Section 3.1.
- Debug and test your program exhaustively
  - The submitted program is expected to compile successfully and run. If the program does not run, the grade for that phase is 0. However, a running program with complete features may not necessarily get full credit, as implementation (i.e., code) will still be checked.
  - As part of the requirements, you are required to create a test plan and execute your program against it. See Section 3.2 for more details.

Please also note that MCO2 will look to extend some mechanisms of MCO1; however, MCO2's specifications will be delayed in order to simulate additional mechanics or modification requests by a client after the development of the base system. Additionally, a graphical user interface should be implemented only for MCO2.

3 Documentation

3.1 Internal Code Documentation

You are **required** to include internal documentation in your source code. At the very least, there should be an introductory comment and a comment before every class and every method. This will be used later to generate the required External Documentation for your project. It should follow the Javadoc style guide.

3.2 Test Plan

As part of the requirements of this project, you are also required to submit a test plan document. This document shows the tests done to ensure the correctness of the program.

- It should be presented in a table format. Refer to Appendix B for an example.
    - The test plan template can be accessed in this link. Refer to Appendix C for the guide on how to export the template as a PDF.
- There should be **at least three test cases per function** (as indicated in the Test Description).
    - There is no need to test functions that are intended for interface display/design.
- Test descriptions are supposed to be unique and should indicate classes/groups of test cases on what is being tested.

4 Deliverables

4.1 Submission & Demo

- **The deadline for MCO1 is June 28, 2025 (Sa) 2100 via AnimoSpace.** After the indicated time, the submission page will be locked and thus considered no submission (equivalent to 0 in this phase).
- MCO1 demo is in the form of a recorded video demonstration
    - While you have the freedom to conduct your demonstration, a demo script will be provided closer to the due date to help with showing the expected functionalities.
    - The demonstration should also quickly explain key aspects of the program's design found in the class diagram
    - Please keep the demo as concise as possible and refrain from adding unnecessary information
    - No demo means 0 for this phase

4.2 Deliverables Checklist

The following are the deliverables for both MCOs:

- ☐ Upload via AnimoSpace submission a zip file (see right on how zip file content is organized) containing the following:
    - ☐ A signed declaration of original work (refer to Appendix A)
    - ☐ Class diagram (exported as PDF)
    - ☐ Javadoc-generated documentation for proponent-defined classes with pertinent information
    - ☐ Source code
    - ☐ [MCO1 only] Video demonstration of your program
- ☐ Email this zip file as an attachment to YOUR own DLSU email address on or before the deadline (the video demo may be excluded from the ZIP file due to file size limitations)

5 Collaboration and Academic Honesty

A student cannot discuss or ask about design or implementation with other persons, with the exception of the teacher and their groupmate. Copying other people's work and/or working in collaboration with other teams are not allowed and are punishable by a grade of 0.0 for the entire CCPROG3 course and a case may be filed with the Discipline Office. In short, do not risk it; the consequences are not worth the reward. Comply with the policies on collaboration and AI usage as discussed in the course syllabus.

6 Grading

For grading, refer to the MCO rubrics indicated in the syllabus.

Acknowledgments / Disclosure

The following Generative AI tools were used to assist in the making of this document.

Google Gemini

- The core idea for this project originated from the instructor. Gemini was used to generate the majority of Section 2 of the document. The instructor reviewed and further refined the AI-generated content.
- Gemini was used for generating the list of class options and abilities (Appendix D). The instructor reviewed and adjusted some of the AI-generated content.

ChatGPT

- ChatGPT was used mainly for streamlining and/or sentence construction. The instructor validated the AI-generated output and modified it as needed.

Appendix A: Declaration of Original Work Template

Declaration of Original Work

We/I, [Your Name(s)] of section [section], declare that the code, resources, and documents that we submitted for the [1st/2nd] phase of the major course output (MCO) for CCPROG3 are our own work and effort. We take full responsibility for the submission and understand the repercussions of committing academic dishonesty, as stated in the DLSU Student Handbook. We affirm that we have not used any unauthorized assistance or unfair means in completing this project.

[*In case your project uses resources, like images, that were not created by your group.*] We acknowledge the following external sources or references used in the development of this project:

1. Author. Year. Title. Publisher. Link.
2. Author. Year. Title. Publisher. Link.
3. Author. Year. Title. Publisher. Link.

By signing this declaration, we affirm the authenticity and originality of our work.

Signature and date	*Signature and date*
Student 1 Name ID number	Student 2 Name ID number

[*Note to students: Do not submit documents where your signatures are easily accessible. Ideally, submit a flattened PDF to add a layer of security for your digital signatures*]

Appendix B: Test Plan Format Sample

MyClass						
Method	**#**	**Test Description**	**Test Input**	**Expected Output**	**Actual Output**	**P/F**
getAverage	1	arr contains only non-negative numbers	arr = {1,2,3,4,5,0} arrSize = 6	2.5		
	2	arr contains only negative numbers	arr = {-1,-2,-3,-4,-5} arrSize = 5	0.0		
	3	arr contains a mix of positive numbers, negative numbers, and zeros	arr = {-1,0,5,-4,7,1} arrSize = 6	3.25		
	4	arr contains only zeros	arr = {0, 0, 0} arrSize = 3	0.0		
anotherMethod	1	**...**	**...**	**...**		
	2	**...**	**...**	**...**		
...	**...**	**...**	**...**	**...**		

Given the sample method above, the following are 4 distinct classes of tests:

1. Testing with arr containing only non-negative numbers
    - Non-negative numbers (positive numbers and zeros) are the only values that can be part of the average computation
2. Testing with arr containing only negative numbers
3. Testing with arr containing a mix of positive numbers, negative numbers, and zeros.
4. Testing with arr containing only zeros
    - Zero is a non-negative number

The following test descriptions are **incorrectly** formed:

- **Too specific:** Testing with arr = {1, 2, 3, 4, 5}
- **Too general:** Testing if the function can correctly compute the average of the non-negative numbers
- **Not necessary (because already defined in the pre-condition):** Testing with a negative arrSize

The test plan template can be accessed in this link. Refer to Appendix C for the guide on how to export the template as a PDF.

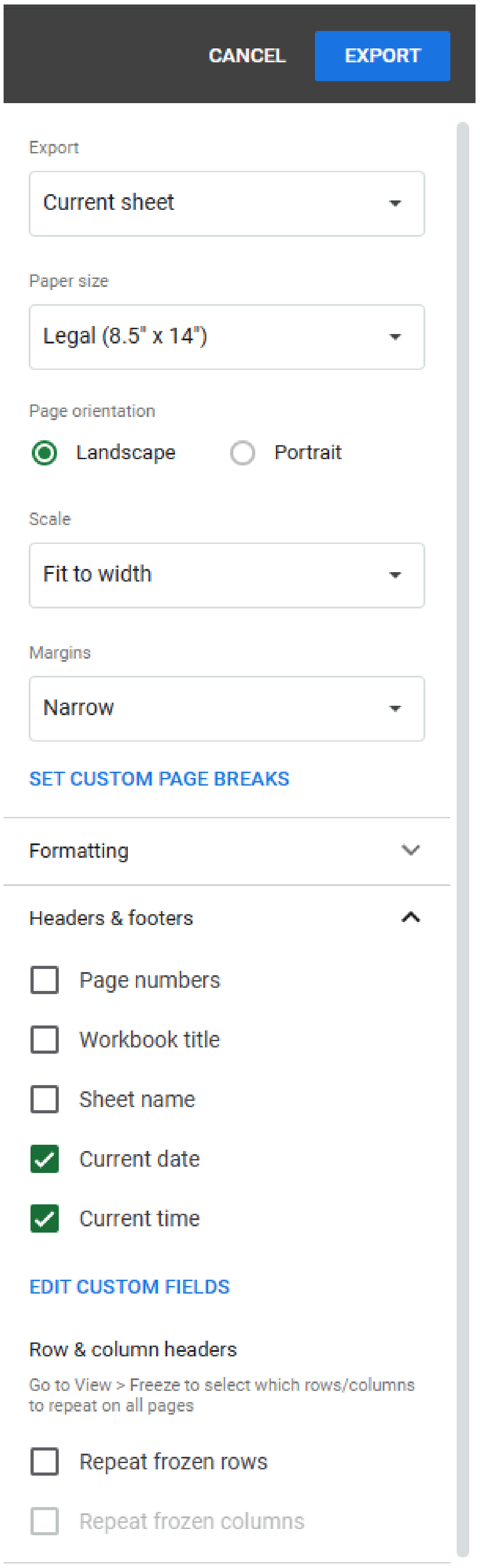
Appendix C: Export Test Plan Guide

1. In the **TestPlan** sheet:

   File > Download > PDF (.pdf)

2. Adjust the settings to:
   - Export: Current Sheet
   - Paper size: Legal (8.5 x 14)
   - Page orientation: Landscape
   - Scale: Fit to width
   - Margins: Narrow
   - Formatting dropdown
     - Leave as is
   - Headers& footers dropdown
     - Check: Current date
     - Check: Current time
     - Uncheck: Repeat frozen rows

3. Click Export.



Export
Current sheet

Paper size
Legal (8.5" x 14")

Page orientation
[✓ Landscape] Portrait

Scale
Fit to width

Margins
Narrow

Formatting

Headers & footers
- [ ] Page numbers
- [ ] Workbook title
- [ ] Sheet name
- [✓] Current date
- [✓] Current time

Row & column headers
Go to View > Freeze to select which rows/columns to repeat on all pages
- [ ] Repeat frozen rows
- [ ] Repeat frozen columns

Appendix D: Class Options and Abilities

Mage		
Mages command arcane energies, specializing in powerful spells and mystical manipulation		
Ability	**Description and Effect**	**EP Cost**
Arcane Bolt	Launch a basic magical projectile that deals 20 arcane damage to the target.	5
Arcane Blast	Unleash a burst of fiery energy, dealing 65 arcane damage to the target.	30
Mana Channel	Draw upon ambient magical energy to restore your own. Restores 15 EP.	0
Lesser Heal	Weave a minor healing spell to mend your wounds. Restores 40 HP.	15
Arcane Shield	Conjure a protective barrier of mystical energy around yourself. You do not take any damage for the round.	12

Rogue		
Rogues are agile and tricky, relying on precision and debilitating opponents		
Ability	**Description and Effect**	**EP Cost**
Shiv	A quick, precise stab that deals 20 physical damage.	5
Backstab	Strike a vital point and deal 35 points of physical damage.	15
Focus	Take a moment to concentrate, restoring your mental energy. Restores 10 EP.	0
Smoke Bomb	Throw a smoke bomb, making you harder to hit. You have a 50% chance of evading any incoming attacks in the current round.	15
Sneak Attack	You rely on your agility to evade your opponent, taking no damage from any of their attacks, while you deal 45 physical damage to them.	25

Warrior		
Warriors are tough, focusing on direct combat and robust defense		
Ability	**Description and Effect**	**EP Cost**
Cleave	A sweeping strike that deals 20 physical damage.	5
Shield Bash	Slam your shield into the opponent, dealing 35 physical damage.	15
Ironclad Defense	Brace yourself, effectively taking no damage for the current round.	15
Bloodlust	Tap into your inner fury, restoring a small amount of health. Restores 30 HP.	12
Rallying Cry	Let out a powerful shout, inspiring yourself and recovering 12 EP	0