CMP1903M Object Oriented Programming 2024 – 2025

# Assignment 2: Report

[Expand the sections as necessary]

Name:

Student ID:

Code repository URL:

Video URL:

# Application:

1. Reflection on the OO features within your code. (~400 words)
2. Reflection on your handling of error conditions in your code. (~200 words)
3. Reflection on your testing activities: What did you test, and how did you do it? (~400 words)

Eg: I tested the application against ….

1. Include evidence of the tests (screenshots are OK)

# Reflection & Feedback:

1. What was the most important thing you learned from this assessment? (< 200 words) Eg: I learned that If you don't think every day is a good day - try missing a few. You'll see.
2. What was the most challenging aspect of this assessment and how did you approach it? (<200 words)

Eg: I started painting as a hobby when I was little. I didn't know I had any talent. I believe talent is just a pursued interest.

1. What would you particularly like to receive feedback on in this assessment?

# Assignment 2 Checklist

All of the elements in a section must be checked for it to be considered for that grade (this isn’t guaranteed though). All previous elements must also be complete for a grade to be considered.

## Pass standard:

|  |  |
| --- | --- |
| The code compiles and runs. |  |
| The player can explore at least two interconnected rooms. |  |
| Object instantiation, method calls evident. |  |
| Rooms have descriptions and can contain one item or one monster. |  |
| The Testing class is used. |  |
| The player can pick up items and battle monsters. |  |
| Implement at least one abstract class (e.g., Creature) with Player and Monster inheriting from it. |  |
| Use inheritance to define a hierarchy for items (e.g., Item with subclasses Weapon and Potion). |  |
| Demonstrate simple method overriding for Creature subclasses (e.g., different attack methods for Player and Monster). |  |
| Handle invalid commands gracefully without crashing the program. |  |

## 2:2 standard:

|  |  |
| --- | --- |
| Include at least three interconnected rooms with navigation between them. |  |
| Rooms can contain multiple items or monsters. |  |
| Implement at least one interface (e.g., IDamageable) applied to both Player and Monster. |  |
| Use a collection (e.g., List<Item>) to manage the player’s inventory. |  |
| Allow players to view and use multiple items. |  |
| Implement dynamic polymorphism (e.g., items like Potion and Weapon have different effects when used). |  |
| Use LINQ queries for at least one task, such as filtering inventory items or sorting monsters by strength. |  |
| Handle more complex errors, such as trying to attack a monster in a room with no monsters. |  |
| Method calls from ‘Main’ to methods in other classes |  |
| Class definitions show encapsulation. |  |
| A Statistics class is used |  |

## 2:1 standard:

|  |  |
| --- | --- |
| Include at least five interconnected rooms with varied challenges (e.g., some rooms have locked doors requiring a key). |  |
| Monsters have different levels of difficulty and attributes (e.g., health, strength). |  |
| Implement inventory management that supports item limits and item removal (e.g., discarding or using items). |  |
| Use LINQ for multiple inventory-related tasks (e.g., finding all healing items or the strongest weapon). |  |
| Implement at least two interfaces (e.g., IDamageable, ICollectible) and apply them appropriately to classes. |  |
| Demonstrate advanced inheritance with deep class hierarchies (e.g., Monster subclassing into Goblin, Dragon, etc., each with unique behaviours). |  |
| Use polymorphic methods for both items and monsters. |  |
| Provide clear feedback to the user for invalid actions (e.g., “You can’t attack; there’s no monster here.”). |  |
| Testing class uses verification methods in code (such as debug.assert()) to check code. |  |

## First standard:

|  |  |
| --- | --- |
| Use of virtual/abstract methods |  |
| protected access control is used in class hierarchy) |  |
| The Testing class implements a way to record testing data (through a log file for example |  |
| Create a fully navigable game map with at least seven interconnected rooms, including special rooms with unique challenges (e.g., puzzles, traps). |  |
| Add randomness to gameplay (e.g., monsters or items appear randomly in rooms). |  |
| Allow the player to win or lose the game based on health, inventory management, or defeating a final boss. |  |
| Implement and justify design decisions (e.g., why specific classes use inheritance vs. interfaces). |  |
| Use LINQ extensively, such as filtering, grouping, or sorting items and monsters. |  |
| Allow players to interact dynamically with inventory (e.g., sort by item type, use the strongest weapon automatically). |  |
| Demonstrate dynamic polymorphism through overridden methods for both items and creatures. |  |
| Use static polymorphism where applicable, such as overloaded constructors or methods. |  |
| Include a scoring system or player progression (e.g., levelling up after defeating monsters). |  |
| Implement save/load functionality to persist game state. |  |
| Add a basic AI for monster behaviours (e.g., monsters may attack first or flee when weak). |  |