CMP1903M Object Oriented Programming 2024 – 2025

# Assignment 2: Report

[Expand the sections as necessary]

Name: Muhammad Farhan

Student ID: 29100057

Code repository URL: <https://github.com/Syed-MAF/assessment-1-.git>

Video URL:

# Application:

1. Reflection on the OO features within your code. (~400 words)

Within my code I have used many object-oriented programming features like instantiation, method calls, encapsulation, abstraction, inheritance and polymorphism. Instantiation is used when objects have been created in the game with the required data that they need for example player has a name and health, items like weapon has a name and the damage that it does and potion that has a name and the health it gives to the player.

Method calls have been widely used everywhere in the code to make the code simpler and to make the game easier to write for example any whenever we need to check the players health we do not have to write the same block of code again and again, rather we can make a method that outputs the players health and whenever we need to check the players health we just call the method. This ensures that the code for the game remains easy to code and simple to understand.

In the code abstract classes have been used for example the Player and Monster classes inherit from the class Creature, both Monster and Player can use the method TakeDamage(), however the method PickUpItem() is only in the Player class as monsters can’t pick up items.

Furthermore, this shows that inheritance has been used within the code as a hierarchy was created when Player and Monster inherit from the Creature class as said before. Also, subclasses were used such as in Item where subclasses were made for weapon and potion. The class hierarchy is player and monster inheriting from creature and goblin and dragon inheriting from monster.

I have also used encapsulation in classes to encapsulate data and functions so that it can protect an object’s state by controlling access and exposing only the necessary functions causing the code to be more secure and maintainable. An example of where I have used encapsulation is in player.cs as to use the inventory, methods like PickUpItem() and UseItem() have to be called to interact with the inventory, which does not affect other code.

Polymorphism has been used in the code as it allows parts of the code to use a shared interface to make the code simpler so for example it doesn’t matter whether it’s a goblin or a dragon attacking for the attack to happen shown in the line currentMonster.Attack(player). Additionally goblins and dragons have different attack behaviour as they cause different damage to the player. Items in the game also have unique effects as potions heal the player but weapons cause damage to the player.

1. Reflection on your handling of error conditions in your code. (~200 words)

To handle errors in the code I have used exception handling by using try-catch to find the errors before it crashed the code for example have used the exception handling over all the options that you can select in DisplayRoomOptions(), as the message shown will show which part of the code cause the error and the code to crash, meaning fixing the error will be much faster. Also, throughout the whole code I have had to manage errors like user inputs, so making sure the user does not type anything they want and only selects one of the options. For example, when asking to select an item to use the program still continues as normal even if the user inputs an item that isn’t in the inventory. Also, another example is making sure that the player fights the monster in the room before they are allowed to go to the next room and when I used string.IsNullOrWhiteSpace to ensure that the user does not enter a blank name.

1. Reflection on your testing activities: What did you test, and how did you do it? (~400 words)

Eg: I tested the application against ….

I created the Testing class to run the test in my game to make sure that the key methods to run the game was working. I used the debug.assert method which ensures that the code behaves as it should. It also allowed me to see where my code is making an error by printing a message that tells me what the problem is. An example of how I used the testing method is when I created a new object player and the objects weapons to test if the player is able to pick them up. I made the player pick up the sword- player.PickUpItem(weakSword); and used debug.assert here- Debug.Assert(player.EquippedWeapon != null, "Player should have a weapon equipped");.

What this line of code does is that it checks that the first argument is true, and if it is not true is displays the message saying ’player should have a weapon equipped’ which tells me that the test didn’t work as the user hasn’t equipped the weapon. If the test works then the method runs as normal so you can see in the screenshot that the message outputted when these two lines , 1.Debug.Assert(player.EquippedWeapon != null, "Player should have a weapon equipped"); 2. Debug.Assert(player.EquippedWeapon.Name == "Wooden Sword", "Should equip first weapon"); , give this output.

A screenshot of a computer

AI-generated content may be incorrect.

This means that the methods works as they should and because the first statements in the debug.assert are false the output message is not printed which only prints if the method did not work as it should.

Just like I explained now, I also tested using debug.assert, if the stronger sword is automatically equipped, if the monster inflicts damage on the player by testing the Attack() method, if the player is able to pick up an item by using the method PickUpItem(), and if the item is found in the inventory once it is picked up.

I have used the testing class to check the basic things that allow the game to run are working as they should, however ive learnt that if I used the testing class as I was writing all the methods I could have been able to test the whole program at a much larger scale making sure each method worked. This would also allow me to find errors withing my code as I am writing the code saving lots of time at the end.

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

A screenshot of a computer

AI-generated content may be incorrect.

# 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.

Throughout the assignment I’ve learnt countless skills however I believe one of the most important skills out of those is learning how to use tests in my program and how to use the debug.assert method and how you can test you code as your building it which allows you to fix errors as they are made saving lots of time trying to fix issues at the end. It also means that you are able to know which specific part of the code is causing issues which means fixing the problem is much faster. I also learned how to use inheritance and abstraction in classes which makes writing code much more simpler as you are able to reuse the code from another class rather than having the same block of code somewhere else. It also means different classes can have the same methods which is very useful as in this game both player and monster can attack using the Attack() method. I also learned how to use LINQ and how it makes code much shorter and efficient rather than having to make write a block of code of if-else statements.

1. 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.

The most challenging part of the assignment was making all the methods in the different classes, and I built the program as I had to learn the basics of object-oriented programming all over again as I was struggling and tried to implement them in this assignment. After practice I got the hang of using different classes and having different methods in them which ran the whole game. I also had to learn how to use debug.assert as at first, I did not understand it at all however I realized it was actually really easy to use. If I had used the testing class from the start, I would have been much easier to understand how to use testing rather than trying to add it at the end.

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

I would like to receive feedback on how I can make error handling much better throughout the code and if I have used object-oriented programming features correctly throughout my code.

# 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 behaviors). |  |
| 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., leveling up after defeating monsters). |  |
| Implement save/load functionality to persist game state. |  |
| Add a basic AI for monster behaviors (e.g., monsters may attack first or flee when weak). |  |