CMP1903M Object Oriented Programming 2024 – 2025

Assignment 2: Report

Name: William Paul Viziteu

Student ID: 29171215

Code repository URL: <https://github.com/WillPaulViz/DungeonExplorer>

Application:

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

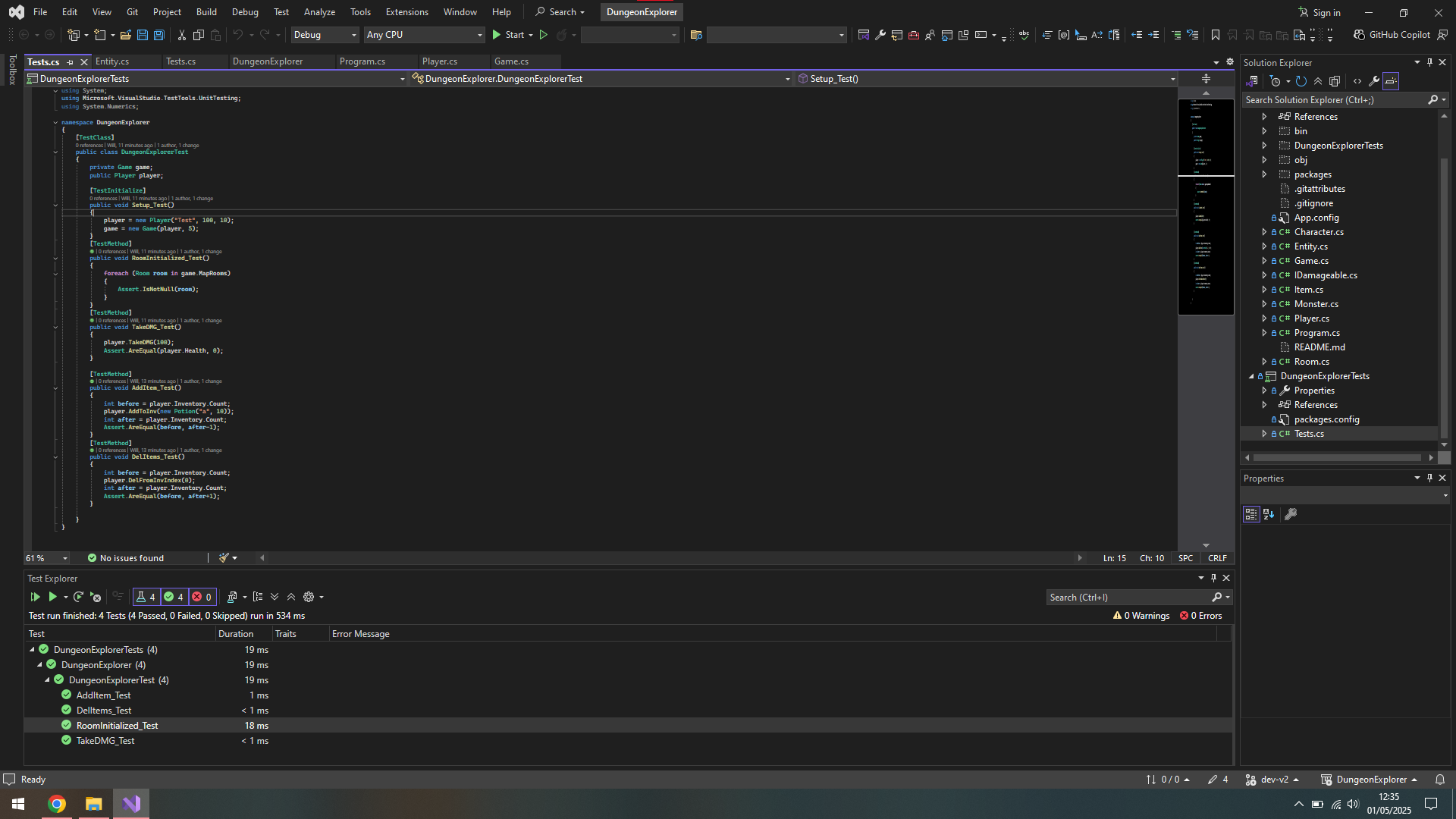
I believe that I have included many of the Objective Oriented Programming features within my code, such as inheritance, polymorphism and abstraction. While programming this project, I came to the realisation that many of the classes use very similar variables and methods. To combat this problem, I created an abstract class that all other classes in the project use as a ‘template’ of sorts, this allowed me to use recursion, so that I could call other classes within classes, and so that I could also clean up my code base in many places. Polymorphism was very tricky to implement into my code, because for the previous assessment in this module, I did not have any classes that inherited from one another or used one another, so figuring out which classes needed type casting, and which needed to be initialized with its parents' classes was difficult. Interfaces were difficult to wrap my head around, firstly their syntax is the same but different from normal, and secondly, I didn’t see a point to interfaces. I was later proven wrong by my own code, after I did some moving of functions around in classes, my compiler told me that I had a missing function in one of my classes, here is when I realised that interfaces were good, because if it didn’t tell me, it would’ve used the original virtual function instead of the overriding function. I did later decide to change how the function works but I kept the interface, because it’s still being used ‘just in case’ something isn’t there. Using LINQ seems very simple, but you do also have to learn the syntax of it which was a bit of learning curve. I can see many places in my code where LINQ would probably be more applicable to use, but personally I prefer writing as much as I can in the language that I am using, even though it is a module for it, it seems like it’s a different way to a process in a different language so I won’t be using it a lot.

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

I do think that I have handled all the error conditions were the user gives an input, but some methods within classes do not have error conditions where you would typically find them. I believe that I have structured my code that even without the error checking conditions for some of those methods, there still will not be any errors shown for them since they either have been handled before or I thought it was not needed. For text inputs in the game, I’ve relied on a function to clean up the string inputs, and for integers they are repeated until there is a valid integer. I do think that I could’ve had an independent function to handle the strings and integers with its own error statements, but in the way I structed this code I implemented it like this instead of rewriting it to make the games output to look a bit nicer.

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

Initially I did not intend to use testing, because in all my years coding experience I have never used testing in these types of projects to create a separate project in the same namespace, and to create separate classes with separate functions, while referencing the other project. I do see benefits of this in large projects, but in smaller projects it doesn’t seem that intuitive, because classes and their methods change a lot, so it is a lot more for the programmer to maintain but it does make them sure that the code works. Testing is good also when you want to see what the of some variables are, and there are many places to test different things in the code, but I didn’t see a need to implement them into my testing since of what outputs they would give me. I tested for the initialization of the game class which checks for if there are any rooms generated in the map object. I also tested if the characters could take damage, by calling the player take damage method, and removing all their health to see if it got set to 0. I then tested for if entities could be added to the inventory list, by adding or removing an item to the player, then comparing the before and after of the length of the inventory. I believe I will use testing like this once the project is too large to test by myself, because at this point, if I make changes in the projects code, there is a chance I would also have to change how the test code checks for something, or even if you tested only specific methods, there wouldn’t be a point if that method will always pass the test, so it isn’t really that maintainable in smaller projects like this in my opinion. However, I do believe that if I did implement the testing from the start of the project, I could’ve had less errors in my compiler and guess work, which would only really help me save time.



Reflection & Feedback:

* 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.

Polymorphism was something new that I learnt, and that it’s an essential when wanting to have flexibility in my project, it let me use different methods depending on the type casing, and to save multiple types of classes in a list. That part was very important for my project and if it didn’t exist, my code base would’ve been a whole lot longer and much more difficult to maintain. This lets me change one function in a class, and it will not affect the other classes

* What was the most challenging aspect of this assessment and how did you approach it? (<200 words)

The most challenging part of the assessment funnily enough was the initialization of the room; I could not for some reason add items to monsters that were in a room, and it took a few hours before I changed the approach of how I went about it, before I was trying to add items into monsters then into the room, but then I changed it to make the rooms, then the monsters, then the items. There wasn’t really one specific part of the assessment that was a challenging aspect.

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

The variable/method naming and structure of the project.

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

2:2 standard:

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