**Game Outline:**

I have designed a game in which the player must battle monsters and solve puzzles in order to progress to the next room. Each monster has a unique difficulty value, this affects their health and their attack damage. The player must progress through 7 rooms in order to complete the game. Each room can contain a weapon and/or a spell. These can be added to the player’s inventory and Weapons can be equipped. The player can use spells to heal their character, which removes the spell from their inventory.

**Inheritance:**

I’ve used inheritance in my code to better control the behaviour of Rooms, Creatures and Items. The Room class is the superclass that PuzzleRoom and MonsterRoom inherit from. Items serves as a base class for Spell and Weapon. Creatures has the largest inheritance hierarchy: Creature is the base class and Monster and Player are both child classes. The Monster class has 5 children, Dragon, Shulker, Skeleton, Warden and Witch that inherit behaviour from Monster. *[show inheritance tree diagram, show this being true in the code too]*

**Dynamic Overloading:**

The Monster class contains a virtual method, which child functions have the option to override to implement their own behaviour. This is an example of dynamic polymorphism or runtime polymorphism *[show screenshot of override method in Skeleton class versus virtual method in Monster class]*

~~As an example of static polymorphism is shown in the two turn decision methods. I use static polymorphism to be able to display two types of instructions to the Console depending on the type of room that the Player is in.~~ *~~[show UserInterface.ShowTurnDecisions]~~*

**Static Overloading**

I have used polymorphism in to display different messages to the console depending on the type of room given as a parameter to the function. *[show UserInterface.ShowTurnDecisions]*. Using polymorphism in this way helps me to reduce the maintenance of code, if my code needs to be changed, it only has to be done in one place rather than in multiple. I think it enhances readability, which helps others understand my code quickly.

**Other Polymorphism Stuff**

I have used polymorphism a lot in my code. Its used to let me iterate over any object that implements the IEnumerable interface, allowing me to generalise my code (rather than creating a new function and logic flow for each type of collector.) *[show screenshot of UserInterface.DisplayEnumerable].* My code also implements an inventory for the player. This is a list of type Item which contains Spells and Weapons, which are subclasses and inherit from Item. This allows me to store various types items in a list whilst maintaining the ability to sort them into their specific types *[screenshot of Inventory.GetWeaponsInInventoryAscending() and Inventory.GetSpellsInInventory()].*

**Error Handling**

Error handling for user inputs is most important in my code. It is important to quickly catch illegal input and ensure that any input meets the requirements of the code. Typically, the user must input an integer to select an action that they wish the Player to carry out. As such, I check whether the value is within a certain range. If it is not, then I let the player know that the input is invalid and how they can comply.

**LINQ**

**Interfaces**

**Virtual/Abstract Methods**

**Protected Access Control**

**Testing Class**