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]*

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]*

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()].*