Dispatch Table Design Decisions

In this dispatch table, there are four classes in total. There is one base class called Fighter and three classes derived from this base class called Player, EasyNPC and HardNPC. I used an object oriented approach which ensures the code is abstract and reusable as much as possible, and it also includes inheritance and polymorphism.

I decided to use a base class called Fighter to derive all fighters from, of which I derived a Player class and two NPC classes. The two NPC classes are both different in difficulty (easy and hard) and they both have different fighter names so that the console output values can be clearly seen for each fighter. A virtual function was used in the fighter class to determine the attack damage value as speed + resistance. This was overridden by the NPC classes to alter the values according to their difficulty, whereas the Player class uses the function without overriding. I used dynamic casting to output the values of speed, resistance, health, and name of each fighter after initialising them to show polymorphism working in full extent.

I decided that the attack function should be virtual because it would be different for each NPC based on their difficulty and to follow the principles of polymorphism, I made sure that the base class had this virtual function. I also used upcasting to output each fighter’s attack damage to the console so that the dispatch table can be verified. It would have been inefficient to create separate functions for each fighter to match their attack damage.

These decisions have been made to strive for the highest levels of implementation of the specification and uses dynamic binding and casting to show off the dispatch table. On reflection, I would make the code more efficient by moving functions to base class or vice versa to maximise the reusability of code.