Design Rationale for Requirement 4

The diagram represents an object-oriented system for player classes, the respective starting weapons, and their special skills.

The three player Classes(capitalised to reduce confusion), Wretch, Bandit, and Samurai are represented as classes of the same name. The Player class has been changed to an abstract class to allow for these Class classes to extend and implement.

The three associated weapons of each class, Club, Uchigatana, and Uchigatana are also represented as their own classes Club, GreatKnife and Uchigatana respectively. They all extend the WeaponItem abstract class, and have a 1 to 1 association from the Class to the weapon to represent that each Class starts with their respective weapon.

The Great Knife weapon's Quickstep skill is represented as its own class, Quickstep, which extends the AttackAction abstract class, as it is a special attack. The 1 to 1 association from the weapon to Quickstep shows that the Great Knife weapons allows its wielder to use Quickstep.

Similarly, the Uchigatana weapon's Unsheathe skill is represented as its own class, Unsheathe, which extends the AttackAction abstract class, as it is a special attack. The 1 to 1 association from the weapon to Unsheathe shows that the Uchigatana weapons allows its wielder to use Unsheathe.

The use of abstract classes to represent the player Classes and weapons adheres to the Open-closed Principle, by not modifying existing code. It allows for new Classes and weapons to be added in the future with minimal risk. It also adheres to the DRY principle by reusing code through inheritance.

A downside to this approach is that the code and logic for player Classes and weapons is spread across multiple classes rather than in one place, which can make it harder to understand. It also results in larger code.

Changes in Assignment 2

The decision to implement the player as an abstract class Player allows future player classes to be easily implemented, by extending the Player class. It also adheres to the DRY principle of reusing code, through inheritance. A disadvantage of this implementation is the code and logic for the player is stored across multiple classes, making it harder to understand.

The relationships between GreatKnife and QuickstepAction, and Uchigatana and UnsheatheAction, are now dependencies.

The decision to implement QuickstepAction and UnsheathAction as their own independent actions allows future weapons which also use the actions to be easily implemented. This also adheres to the DRY principle of reusing code, through inheritance. A disadvantage of this implementation is the code for the special actions are stored across multiple classes, making it harder to understand.

UnsheatheAction now creates a new instance of UnsheatheWeapon which is used as a temporary weapon to pass into the AttackAction, to achieve the x2 damage and 60% accuracy modifiers. A downside to this approach is that UnsheatheWeapon is an additional class which must be implemented, and that the code and logic for UnsheathAction is stored across multiple classes, making it harder to understand.