Designing and Implementing

Designing:

Our current design incorporates composition, aggregation, association, and inheritance relationships. For inheritance, the major classes of Entity, Item, and Location were superclasses to many subclasses, especially the Item class which had a number of subtypes. Aggregation was used quite a bit, such as when an entity exists inside of a location or inside of a combat. Similarly, Composition was used in the same kind of situation except that the contained class only really exists inside of the container class. For example, inventories only exist in the context of a player using one. Other inter-class relationships like realization and dependency were not included as they weren’t taught in class, nor were they relevant to the Genshin Impact system. In terms of the game design itself, we chose to create a text-based choose your own adventure style game instead of the original 3D game. We made this decision knowing that it is unrealistic to create a 3D game with the intended level of complicity given our time constraints.

Implementing:

In the first sprint, we created just enough classes to ensure the basic functionality of the game. Such classes are Artifact, Ascension, Boss, Chest, Combat, Domain, Entity, ExpMaterial, Food, Game, Hero, Inventory, Item, Location, Monster, Player, SpiralAbyss, Talent, and Weapon. Additional classes related to fishing, fate, animals, Serenitea pot, achievements, tutorials, and quests were saved for later sprints. Similarly, only variables that were necessary for the basic functionality of each class were implemented. For example, constellation and friendship from the Hero class were omitted due to their minimal impact on basic gameplay.

Generally speaking, choices made during implementation were based on ensuring that a working game could be created within the given time. Our goals for this first sprint were to create base heroes, locations, a working combat system, and to be able to interact with items. For the next two or three sprints, each of which should be about a week to prevent imbalance in weekly schedules, we plan on implementing a number of things. For the second sprint, these would be the missing classes mentioned earlier, added to create auxiliary content. In more detail, we would like to have functioning fishing and quests systems, Serenitea pots functions which are personally designed locations, interactive tutorials, and achievement and quest tracking. Afterwards, in the third sprint, we would be adding a larger number of base characters and locations so the game has a more significant amount of core content. This essentially means fleshing out the game so it is in a more comprehensive context. Regarding general changes that may happen over either sprint, we would also like to balance combat more effectively and add some kind of progression in conjunction with the other topics mentioned in this paragraph. Perhaps over a longer time period, save files and saved progression would be a target as well.

In terms of dependencies, Location, Item, Combat, and Player classes were depended upon the most. Using the Game class as a conduit for interaction, almost the entire program depended on the Location class. This is because the location class gave the grounds upon which any user input could occur other than combat which required additional context first to make sense. Additionally, the Hero class depended on the Weapon and Artifact classes, which relied on the Item class. These dependent relationships were another factor we considered when choosing classes to implement. Classes that had many dependencies, meaning they depended on and were depended on by other classes, were prioritized during implementation. Classes like fishing and Serenitea pot, with little to no dependencies, were eliminated during this first sprint.

To ensure a working final program, we made sure to effectively communicate during the entire designing and implementation process. By doing so, we were able to omit issues like different variable/method names. It was also decided that the entire program should be made on Replit so that all group members could work on the code simultaneously and collaborate as easily as possible. As a result, we were able to create the game in an efficient manner. Furthermore, this centralized system allowed for a few core classes that acted as a way to test all the others. These were Game which handled the majority of interclass logic by connecting existing methods, and main which called upon game to begin operation.

Class implementation distribution:

Andy: Entity, Hero, Monster, Boss, Player, Combat, Game

Christine: Location, Domain, SpiralAbyss, Chest

Lisa: Inventory, Item, Artifact, Weapon, ExpMaterial, Ascension, Talent, Food