Sprint 3 Design Reflection:

Zeus:

Whilst implementing the Zeus god card looked relatively simple on the surface, there were some nuances that made it somewhat difficult to implement. For example, in the worker class from sprint 2, there was no simple way to get access to the space the worker was currently standing on. This made it difficult to check for the build height of the worker's current position (since we cannot build on a height of 3). To work around this, I made board as an input for worker and made a getSpace function for worker.

Another issue that I encountered was the code smell of having long methods. In the GameController class from sprint 2 there were several methods that were quite long. This made it difficult to navigate through the class, and make necessary changes, to allow the player to build on the space a worker was already standing on. Because the implementation of the function handleSelectWorker automatically deselected the worker when clicked again, I had problems allowing the player to choose to build underneath an already occupied space. Furthermore, because it was hard to navigate through several functions of GameController, it took more time than expected to get my implementation of Zeus to work.

Timer Extension:

The timer function was quite straightforward to implement overall. The main difficulty that arose, was that there was no way to tell the game to finish when a timer ended. This is because the victory screen functionality was embedded within the GameController class, which I was unable to give the PlayerTimer class access to. To fix this I needed to move that functionality into the game class, which the PlayerTimer could then communicate with. However, apart from this small issue, our code from Sprint 2 was easily extensible to fit the timer extension into our implementation of Santorini. This is because it provided a clear structure for how each screen was established, and how it fits into the context of the game and menus.

Helpful Token Extension:

The Helpful Token Extension was simple to implement. Because we followed the open close principle for our Player and Worker class, by having well defined methods, as well as scope, I was able to add a simple capability to add a list of 'helpful actions' to the worker's already existing actions. To do this I simply created another function call generateHelpfulBuildActions, and then in the game controller class I was able to replace these actions with the worker's already existing actions. One thing I could change in the future is to remove some functionality out of the game controller class. This is because the GameController class can be considered as a "God Class" as it has many responsibilities. By being less dependant on this class to carry out game logic, I

would be able to contain most of the worker and player related functionalities in the worker and player classes respectively.