Status Summary

1. **Project Title:** Pac Man Recreation
2. **Group Members:** Max Macaluso, Rohan Suri, Sahib Bajwa
3. **Work Done:** We have made great progress with our project! Thankfully, there is lots of great documentation on this topic (sources included in README.md file) and Python has allowed us to develop quickly. Due to these factors, by the time of submission of Project 5, we should have a roughly playable game. Below is a specific breakdown of work done:
    a. Max: Worked on the classes pertaining to the playable character (Pac man)
    b. Rohan: Worked on the classes pertaining to the enemy characters (Ghosts)
    c. Sahib: Worked on classes relevant to running the actual game itself, in other words, the Driver code/classes
4. **Changes or Issues Encountered:** Our project is largely the same as what was described in Project 4. The only exceptions being to this: unfinished work, method/attribute naming differences, and other small discrepancies
5. **Patterns:** We have successfully been able to implement all patterns mentioned in the Project 4 Design Document. Below are the specifics:
    a. Singleton: Max implemented this in his corresponding code. Although implementing Singleton in Python is not as straightforward as with a language that features keywords *public*, *private*, and *protected*, we were able to successfully implement the Design Pattern by restricting the use of the class's constructor by raising an exception, forcing the client to use the method *getInstance()* instead. This Design Pattern aided in our design by ensuring that only one player instance is ever created, preventing unnecessary error handling and debugging
    b. Simple Factory: Rohan implemented this in his corresponding code. In implementing this Design Pattern, our team believed that Simple Factory was the best approach because of its simplicity and our lack of need for more complicated/advanced features that Factory and Abstract Factory provide. Simple Factory aided in our design in many ways, two of the most important being: (1) separated creational code out from the code that uses it and (2) provided a clean, abstract, and encapsulated way to create new enemy characters
    c. Observer: Sahib implemented this in his corresponding code. Observer helped our design because, without it, the three main classes that we developed would be highly dependent on each other (tight coupling). Furthermore, the Design Pattern gave us the option to implement our own Signal/Slot system that is popular to many game development libraries/frameworks