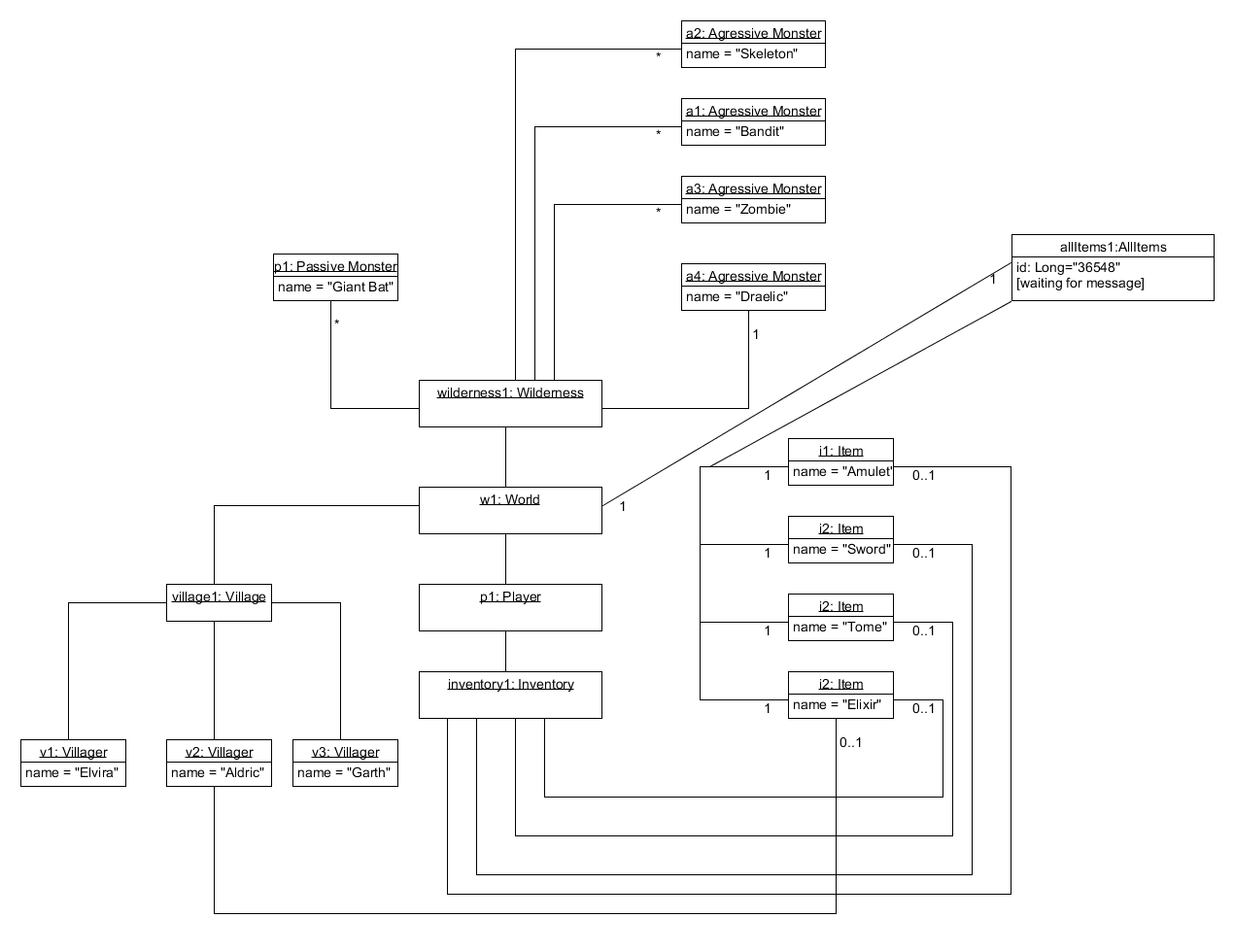
When implementing the game, I found that there were a number of significant changes that I had to make to my game design as it was either more efficient to implement a certain change during the implementation process, or it was needed for certain modules to work properly (i.e Passive Monsters moving randomly around the map, and running away from the player when attacked). First and foremost, when implementing the game, I found that the UML diagram of all the delegation (association) and inheritance (generalization) diagrams that I made in project 2a, were mostly successful in the implementation process as I was able to implement most of these connections between classes easily and successfully on all the aspects of the game. Especially with the inheritance diagram, as I did not have make any new changes to it.

**New Class Diagram made during the implementation of the game:** Note the new relationship between the items, the class AllItems contains all the items, and all the items in the world are maintained here. Also the Wilderness class keeps tabs on both the passive monsters and aggressive monsters rather than having just a monster class. (Allows for easier access)

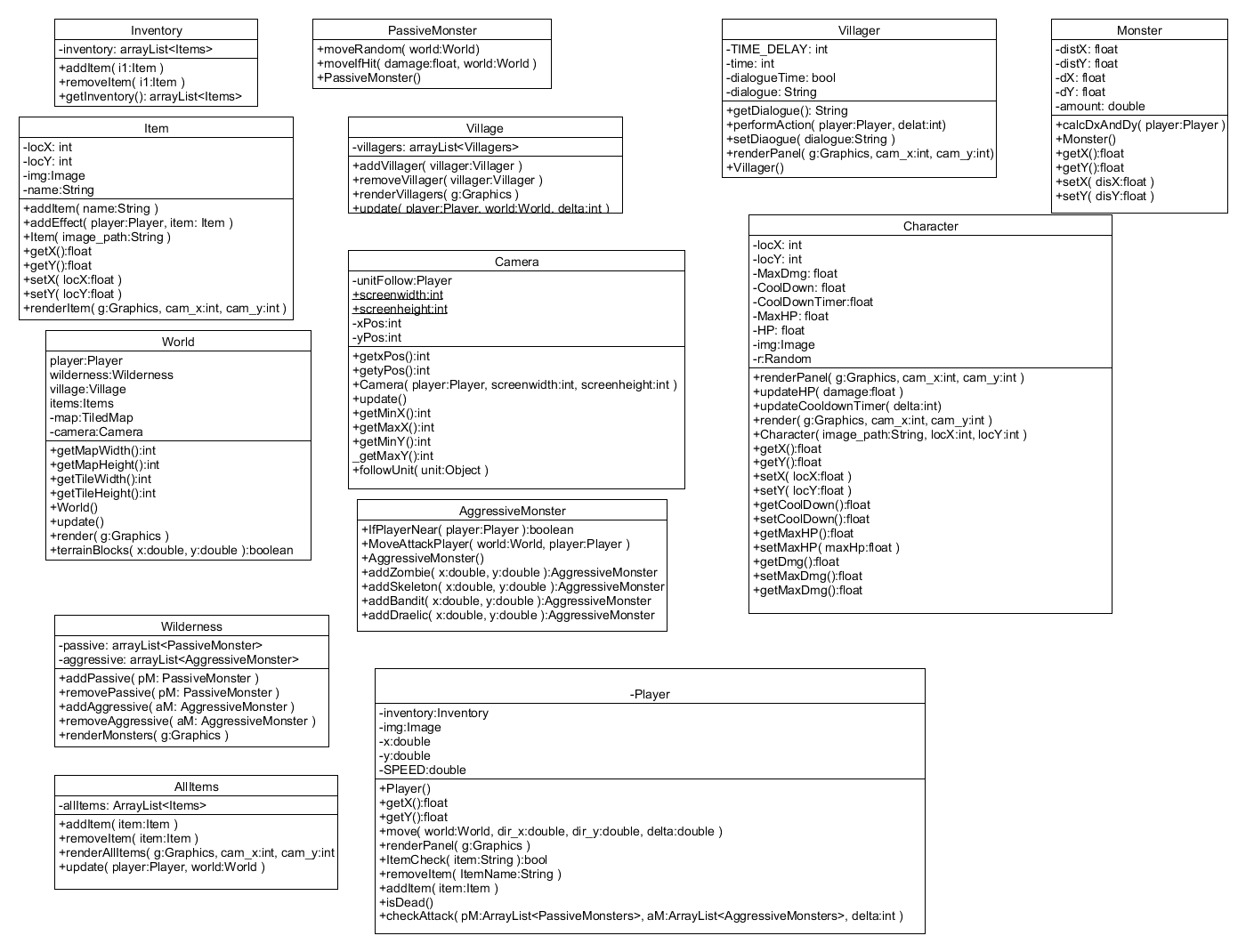
However, there were a few changes that I have made during the implementation between the class relationships as I found it to just be more efficient. For example, in the class diagram above, rather than having the wilderness containing an array of monsters which then extended to Passive and Aggressive Monsters, I choose to have the wilderness contained arrays of both the Passive Monsters and Aggressive Monsters as it was just more efficient to store them here. This allowed me to quickly remove or added specific types of monster more easily, and allowed more control in the monster entities in the wilderness.

Another notable change that I made was that rather than having the world class having four separate items in the diagram, I found it was much better to have the world have a class called All Items, in which in this class, all the items in the world are dealt with accordingly to the current game state. This is similar to what I have done with the wilderness class and monster classes, and the similarly with the village class and villager, as it allowed for more control of what items are currently in the game world, and enabled for an easy update according to the player’s action.

**New object diagram:** Note the new relationship between the items, the class AllItems contains all the items, and all the items in the world are maintained here. Also the Wilderness class keeps tabs on both the passive monsters and aggressive monsters rather than having just a monster class. (Allows for easier access)

Another important and big change that I have made during the process of making the game was the methods and attributes in the classes. As I was implementing the game I realized that there were some attributes that were required for certain parts of the game to function properly, that was not in the old UML diagram. For example, I needed to include a speed attribute for the monster classes, as each monster moved at different speeds (since zombies move slower than bandits etc ). Another example is that I had to include a dialogue timer attribute for each villager as the villager would have to display a dialogue for a specific amount of time.

Also, I had to introduce new methods in certain classes as some of the methods in the old UML diagram alone were not enough or not really efficient in performing certain aspects of the diagram. For example, in the villager class, a new method such as performAction() which would perform the a specific action whenever the player was near, and depending on the player’s inventory. These new changes are displayed in the diagram below.



Throughout the project, there were not many difficulties that I had encountered when constructing the game. The only problems that I encountered were thinking of new methods and attributes within certain classes that would allow certain parts of the game to function problem, as the attributes/methods that I had made in my old UML diagram was simply not enough to carry out certain game requirements. One key thing that I have learnt from this project is that implementations of the first UML diagram, is not necessarily enough to implement the full specifications of the project. Sometimes extra methods and attributes are required to meet all the requirements of the project, and these tend to be added during the implementation process, in order to get certain aspects working. Also, I have learnt that the UML diagram is extremely important as it certainly helps with the implementation process as you have a sort of “blue print” to guide you easily through the project.

If I was given this project again, I would have definitely spent more time on the UML diagram process, and potentially analyse all the possibilities, attributes and methods of how to implement the game. By exploring all the possible class structures and deeply think about certain design decisions, the implementation of the project would have definitely gone much smoother than before. Overall, the project was a long but enjoyable process, as it has taught me how to plan and implement small scale projects. Also, the fact that we were making a game was extremely enjoyable.