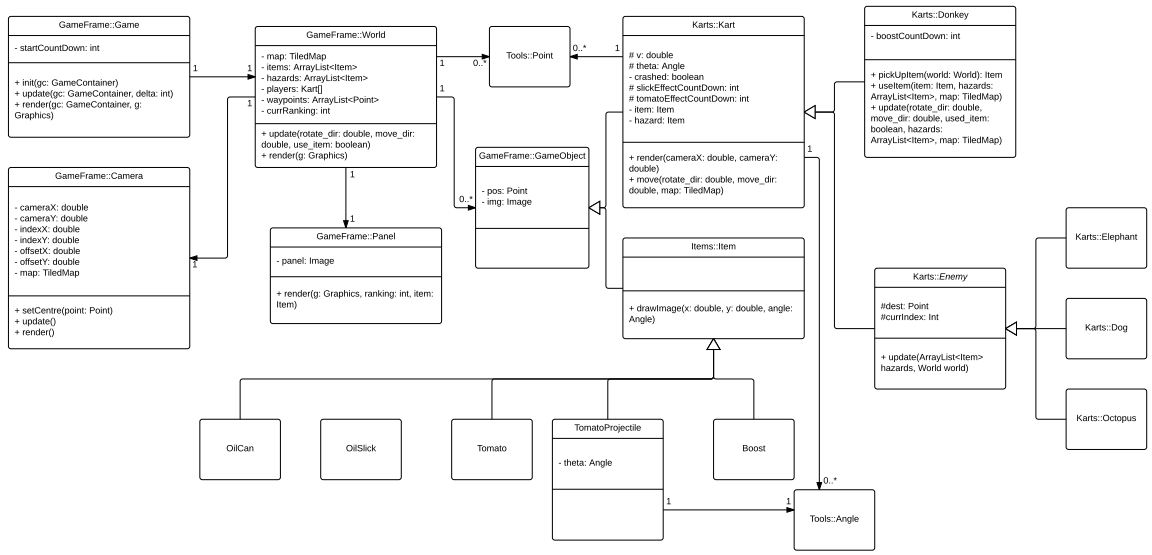
In this reflection, changes I have made on the project, difficulties I encountered and what I would do if I were to do this project again will be stated. The briefed diagram is shown below.



For changes I have made on the project in comparison to the UML at the beginning, the most important aspect that I focused on is encapsulation. For example, I used protected variables without placing classes in a package, which would make no difference with using public visibility.

Moreover, I have made some changes on moving variables from one class to another class due to logical errors. For example, I originally placed waypoints variable in the Kart class as I thought enemies would follow all waypoints. But if I do so, for every kart object I need to read waypoints, which is quite inefficient, and logically waypoints variable is not an attribute that belongs to kart objects, though the benefit is every kart can have its own waypoints to follow. But in this case, reading all waypoints beforehand when initialising the world object would be a better choice and it makes sense that world contains waypoints as they are part of the world.

Then, the biggest change I have made is the about relationships between classes. By modifying the general structure, repetition of codes can be avoided. For instance, I made enemy karts and donkey inherit from the Kart class, and it is obvious that some duplications of variables exist (destX and destY). Though in this case there are only three enemy classes, there can be more duplications when extending the program. Therefore, non-human players were made to inherit from an enemy class containing destination variables.

The last major aspect I made change about is utilising a class to represent variables in classes. The class works just like a user-defined data structure. In my case, I created a class called Point and it contains two variables for x and y coordinates. Then I replaced x and y

variables in classes of GameObject and World, which can make codes more elegant and save more space.

As for the difficulties I met during the project, the first hard thing I met is about the design. At the beginning I was not sure about how to make the structure. Based on the first project I built new classes one by one and tried to look for a proper place to put it in the UML graph. When I got my feedback for project 2a I found there are still issues about the design, and the structure then looks better after I made further changes to stress encapsulation and inheritance.

The second one is the about the waypoints that enemy karts followed. Sometimes enemy karts can get stuck in some tiles, at the beginning I thought my algorithms of movement are incorrect. After checking for many times, I figured out the issue that it is because kart has a rotating speed, when a kart is rotating while moving fast, it travels more distance in the horizontal direction than moving slower. Therefore, enemy karts can finish the game without collision, but it may not if collision occurs.

The last major challenge is how to deal with octopus as it has two states during the game. It is easy to set the octopus to follow waypoints as I can initialise a flag variable indicating whether it is chasing the player. However, the issue that confused me is what octopus would behave once it stopped following the player as we may need to keep track of waypoints it passed while following the player. As it can be quite difficult to track as player can bring forwards and backwards, I chose to compute its nearest waypoint every time it stopped following the player.

From this project, the most important aspect I learnt from it is designing. The implementation is quite easy as long as the design is carried out. But for the design, I asked to myself every time when I wanted to create a new class if the class is extendable and flexible. Such as when I finished camera class, I wrote the constructor as “camera()”, I asked myself what if I want to focus the camera on other players, then I changed it to “camera(player)”.

The other aspect I learnt is the understanding of relationships between classes. Different classes should handle their own attributes within the class methods and avoid too much interactions with other classes but only to pass attributes or derived attributes to other classes.

If I were to do this project again, I would consider more about the design before I started working on the implementation, as I spent too much time modifying the code after I finished the code of previous designs. In terms of the structure, I think the hierarchy is well designed, but for the implementation I would try to reduce some functions to keep the codes easier to modify and understand.