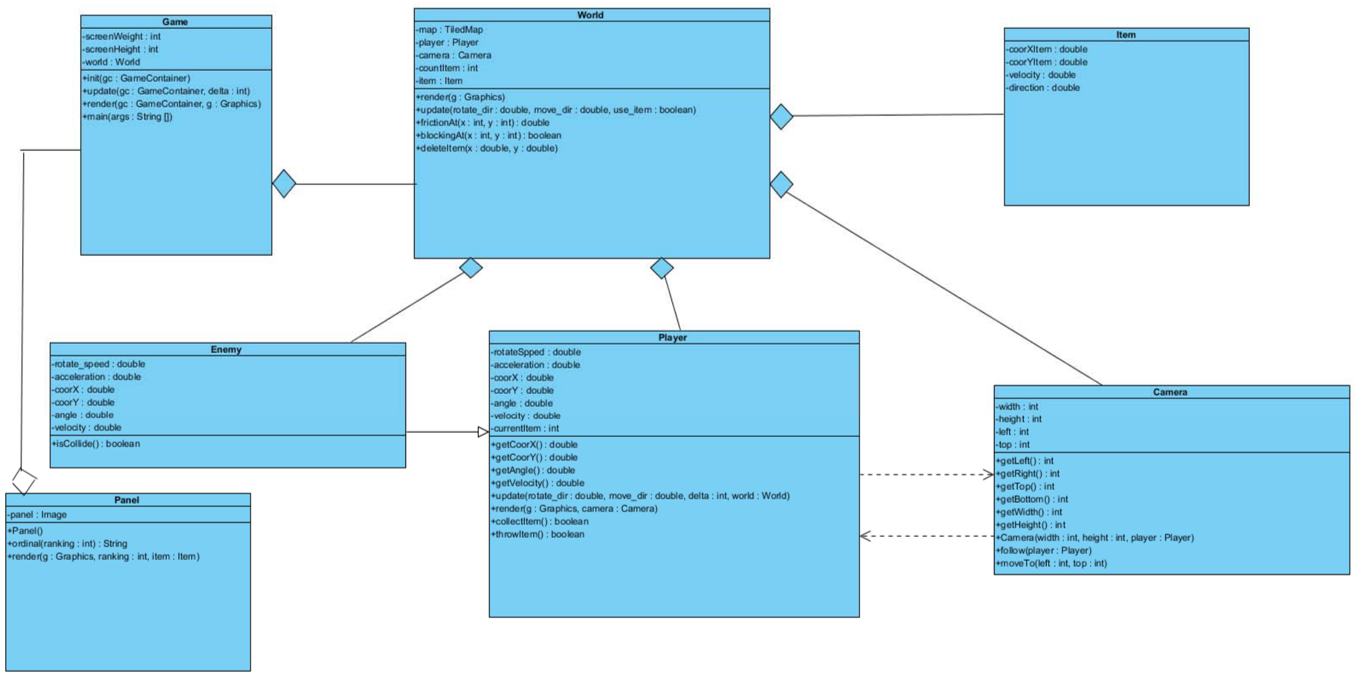
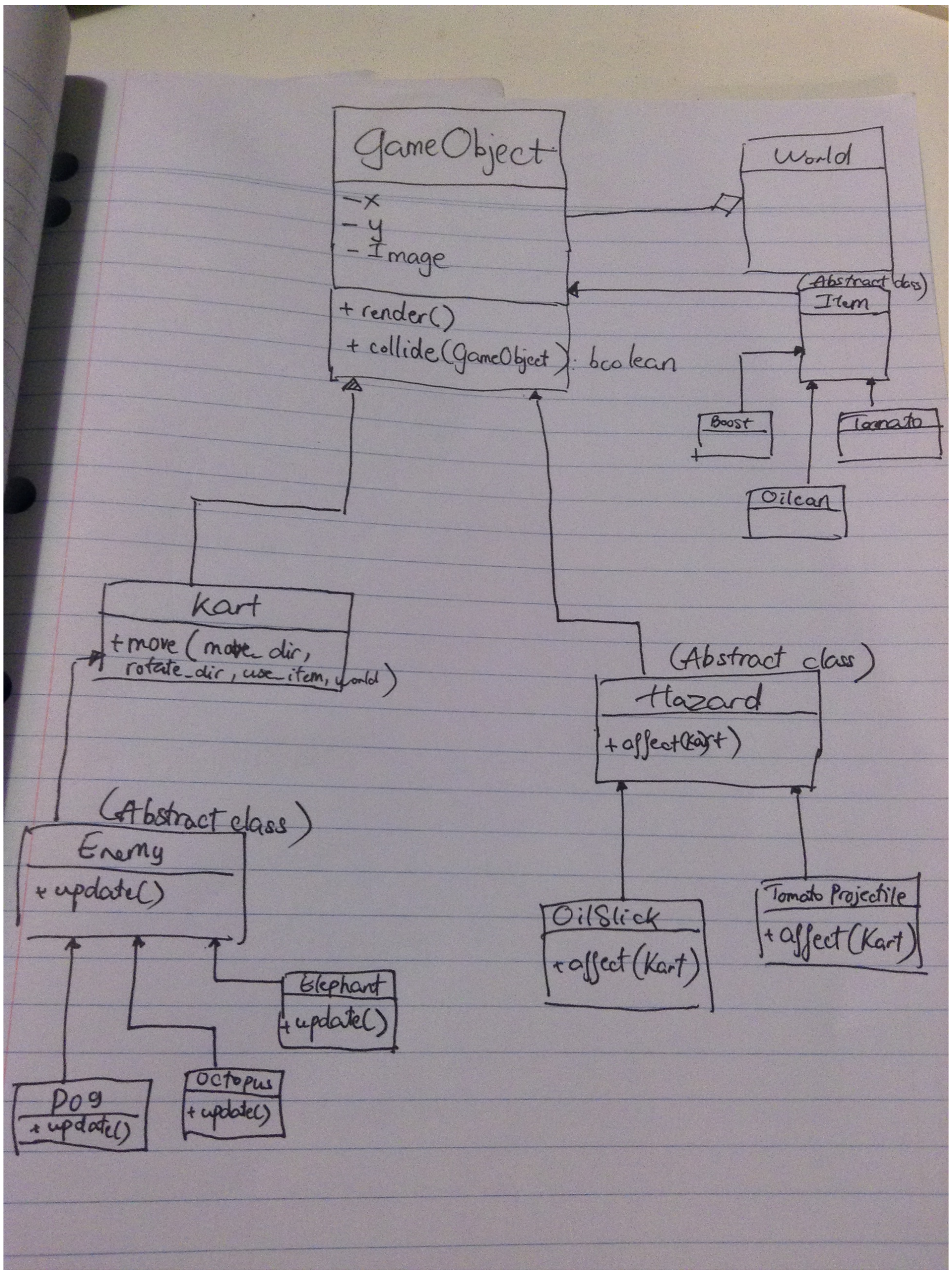
The original game design



It can be seen that from the original game design, Enemy is subclass of Player (because there are many common features between these 2 including x, y coordinate, angle, image, update method…). However, each enemy (elephant, octopus, dog) has different strategy, therefore the update method must be implemented differently, so the update method should be overridden many times. Player can pick up the item and enemy cannot, keep overriding update method is not a good strategy to implement the game, instead creating abstract classes.

New brief game design (doesn't contain Camera and function and its parameter details, focusing more on the structure of GameObject and its child classes and abstract method)



Game Object is the parent class that has the most child class inherited from. Child classes from GameObject include Kart, Hazard, Item. Game Object contains most common features between Item and Kart: x, y coordinates, image, render and collide method. From Kart, break it down to Enemy and Player subclasses, share method move, since both Enemy and Player belong to kart and are able to move.

Put all of the kart into an array (cause we know there’s only 4 karts no more no less, else we can use list as well) so every time they need to be updated or render, just loop though the array and update/render every kart, instead of update/ render each thing individually.

For collide method, if there’s no looping, there will be a lot of if statement (and that’s obviously not good).

When you need to adjust anything, you don’t have to override a lot, need to adjust that one class (or delete or add details, don’t have to depends too much on parent class if use abstract class => more structured and clear when it comes to more implementation or more objects). Abstract classes define abstract methods which the inheriting subclass must implement, provide a common interface which allows the subclass to be interchanged with all other subclasses, the structure will be more clear and neat.

_ The most time consuming part in the project is making the kart follow the waypoints, since the rotate direction depends on waypoint angle (from the elephant), angle of elephant itself and 4 different quadrants, the rotate direction is different for every cases (the ideas of shifting the angle of waypoint from the elephant as well as following the convention are quite complicated even that is not the main point of the project).
Being unable to exploit the Angle class completely is a huge disadvantage (I didn't use the angle class before and came up with all of the complicated coordinate calculation myself, as can be seen from the enemy update code, that took 3 days for me to implement the update method for elephant alone when others who made a good use of angle class took less time, yet if I change it, I need a lot of adjustment).

From the original design, there are only association and inheritance, none of abstract classes are used, leading to overriding many times. With new (partly) game design, 3 abstract classes are created (Enemy, Hazard, Item) and every class inherits from that class can just do what they’re meant to do, and if there’s any adjustment, that one leaf class will need to be adjusted, not changing the whole structure.

One difficulty I found in the project is that there are so many classes that sometimes with so many association/delegation between classes, I was sometimes confused what parameter to pass in a method or where to put a part of code at (even when I chose to use the pickUp item in world.update() or player.update() it both worked the same way, but putting it in player seemed to be more structured), cause world shouldn't handle anything more than update, render, blocking…). Each class should have its own functionality. After I drew a graph, things became clearer.

Another difficulty I encountered is octopus movement. I understand that when it is in player range [100,250], it will treat player as a waypoint, when it’s out of that range, it will follow the normal waypoint. I implemented exactly how it required (even I had to repeat a chunk of code from elephant cause when I simply changed waypoint coordinate to player current position, octopus behaved strangely), then there is a problem. After octopus tries to follow and crash to player for a while (it’s closer to waypoint 3 for example and should head towards waypoint 3), because of the loop, octopus goes back to 1st waypoint and follow from there, doesn't seem to move on after it’s out of the range. If octopus surpasses the player at first, it seems to behave just fine (cause follow waypoint from the very first start).

If I can do this project again, I will definitely make a better use of Angle class and think of a way to implement octopus without having to repeat the code.