Difficulties:

Project 1

1. Calculate the block movement:
   when calculating friction and blocks I had difficulties realizing that we had to 'predict' the future position of the player and then check for blocks.
2. Rendering images:
   I had difficulties understanding the actual world map and the screen portion to be rendered. Especially converting from tiles to pixels when camera is following the player. Since it is rendering the player on the screen from it's position on the map, the player is at the centre of the screen of all time and each frame it renders a portion of the map.
3. Object oriented programming style:
   It is sometimes difficult to decide how and when to use delegation. I had difficulties deciding whether to put as an instance variable or to pass in as parameters and where to place different functions for it to be polymorphic and extensible.

Project 2

1. Following waypoints:
   I had difficulties figuring out how to make the AIPlayers follow the waypoints. Since the rotate direction is depended on where the waypoint is relative to where the kart is facing. I had difficulties doing the calculations and checks.
2. Duration of the effect on kart's movement:
   I had difficulties updating the kart's movement when they are affected by items. I did not know how to calculate the duration and I did not realize to separate the
3. Use of inheritance and delegation:
   Sometimes I had difficulties using inheritance and delegation to simplify the code and make it polymorphic. I had a lot of if-else checks and duplicate code. Eg, when implementing different movement strategies and different hazard, I had difficulties separate calculations and updates.
4. Interaction between hazard, item and karts:
   I did not realize to create a separate hazard class to implement the effect of projectile and oilslick. Also, since they are all very interrelated I had difficulties deciding where to implement which properties and effect.

Knowledge learnt

1. Use of object oriented paradigm:
   I understood the power of delegation, inheritance and abstraction. It allows you to simplify your code and make your functions reusable, more polymorphic and extensible. I was able to use my previous code/functions or make small changes to add more functionality. Furthermore, I understood the use of visibility moderator.
   Eg. Reusing collides and check blocks functions for items and hazards.
   Eg. Using inheritance to have different movement strategies, hazard effects and account for similar type of properties such as x, y coordinates, images and render function.
2. Understand the game programming concepts:

I learnt some basic game programming concepts including user interface, AI updates, back end calculation and display. I grasp the idea of updating in terms of frames per second and also the concept of rendering images on the screen and making calculations for games. Also, I understand how to use framework and extend existing functions and classes to make an application.

Things to do differently

1. Thorough and more detailed planning:

   The use of UML became very important during the implementation stage of the project. I was able to use the partial sample UML solution to implement my game more efficiently and smoothly. Since there are a lot to account for, a better design would speed up the implementation stage.
2. Commenting along the way:

   Since I have not been commenting along the way, I had go back and re-interpret my code and check for bugs which wasted a lot of implementation time. Furthermore, commenting at the end is a huge amount of work and sometimes I could not properly comment without going back and check how my code works.
3. Use debugging tools

Changes in design

1. GameObject class:

   I had to create another GameObject class where kart, item and hazard are inherited from. Since there are similar properties and functions that can be shared between kart, item and hazard, it is simpler to group them into a superclass. It reduces duplication of code and can be more extensible since other type of game object can be added in for future use.
2. Hazard class:

   Another hazard class to account for the effect of the items when kart is hit. This allows simpler implementation and better extensibility. For future changes, items may have different hazards or hazards can be created without items.
3. Collision function:

   I put in another parameter kart for my collision function since for the kart to collide a specific point, the kart calling the collides function will always collide with the point. Therefore the extra parameter is to make sure the kart will not collide with itself. Also the function will return a kart object which allows me to check which kart is affected by the hazard.