Homework 1 for CS202
Bradley Fallon
bfallon@pdx.edu
4/29/2019

DESIGN ANALYSIS

-Overview

My implementation of the drone race involves a hierarchy of classes used to manage the interaction of contestants with the course.

The high level classes are the Race, and the Contestants. The contestants are derived from a hierarchy of Drone related classes. The Race is derived from the Course which is a weighted graph implemented with an array based adjacency list table. The Course graph is a network of routes and waypoints.

-The Race

The Race is the highest level class which manages everything else in the race event. The Race contains the lists of Contestants and tracks their state in the event. The race is derived from Course. The Course is the actual graph of waypoints that the Drones must pass through, but the Course in itself cannot interact with the Drones that are participating. The race is what manages whether a drone is qualified to pass through an obstacle and determines if a drone can pass through the target waypoint. The Race also keeps record of the overall state of the event and the final rank of the Contestants.

-The Contestants

The Contestants are the highest level of derivation in the Drone hierarchy. A contestant is a Controller, and a Controller is a Drone, and a Drone is derived from the commonly used Location base class which is also used in other hierarchy branches. The Contestant is what directly interacts with the race, to track the identification and make decisions about what routes to take and how to maneuver obstacles. The parent of Contestant is Controller, which is what sends the navigation commands to the Drone. The parent of Controller is Drone, which contains the basic information of the drones location and name and behavior for motion. The Drone is also the parent class for DronNode, which is used to track a drone in the race.

The Drone is very basic and adds very little functionality over a Location, but If I had more time to extend this program, I would give the Drone characteristics that affect performance.

The DroneNode is special in that it has one public property. This property is constant and can only be set at construction time. The DroneNode has a public Id number because I see the identification as public information, but it is still protected by being a constant so it cannot be edited by anybody.

-The Course
The course is a graph of Waypoints. Waypoints are virtual, and the vertex array of waypoint pointers contains pointers to many object data types. There are plain Waypoints, which must simply be passed over in order to satisfy the Race requirements. There are then various Obstacle derivations. There is a virtual Obstacle base class, which is derived from Waypoint. All Obstacle derivations are required to override the pure virtual methods that are used to check if a Drone is allowed to pass the Obstacle. There is a required is_completed() method which will return “true” if the obstacle has been successfully maneuvered. There is also a required is_failed() method which returns “true” if the Drone is to be disqualified from the race.

Since the vertex array contains pointers of various types which may or may not be obstacles, it is required to use dynamic casting in order to determine if a given waypoint needs obstacle requirement checking.

-Conclusion
This turned out to be an assignment with a very large scope. I think it would take me months to complete is to real satisfaction, but we are not allowed that kind of time. If I had more time, I would make the Drone have more state variables, such as velocity, and the Obstacles would have more intricate requirements. Currently, the obstacles always pass successfully because I am struggling to find time just implementing the graph and class hierarchies. The actual mechanics of the obstacles seem to be lowest priority when compared to making the Race event able to create the Drone data structures.

There is limited error handling, and this program would certainly need a lot of quality control if it were to actually be shared with a customer.