M2: Group Design

Nate Jones

Design Presentation:

The goal of this enhancement is to provide a user of our simulation software with the capability to give a single command to the simulation that a collection of ships would then follow. The goal of this enhancement is something that is well handled by the Composite design pattern, described in the Gang of Four's *Design Patterns* book. The composite pattern provides a basic template to build off of, but one of the important decisions that was made was where to place the Component interface in our existing class hierarchy.

The way that the Composite pattern was implemented in this project was to insert the Component interface between the Ship class and the Sim_object class in the class hierarchy. If we had inserted the Composite interface as the base class just above the Sim_object class, then we would have to consider Islands to be part of our groups, as Islands are also derived from Sim_objects. This does not make sense for our simulation, as you cannot even give an Island a command in our current simulation. In addition to this, if we had placed it just after the Ship class, then the Composite class would be inheriting Ship functionality from the Ship class, such as the ability to move to different locations, having a fixed position and speed, etc. It doesn't make sense for a group of more-or-less unrelated ships to have a position or speed that is unrelated to that of its members. A group of ships is not a single ship.

Therefore, the Component class was placed just between the Ship class and the Sim_object class in the design hierarchy. This class is called Ship_component, and can be seen in the Class Diagram attached to this document derived from the Sim_object class. The Composite class is implemented by the Ship_group class which derives from the Ship_component class, while the Individual is implemented by the original Ship class which also derives from the Ship_component class. A Ship_group can contain all of the different kinds of Ships and Ship_groups since children in a Ship_group are pointers to Ship_components.

Since we want to be able to use Ship_groups in place of Ships in Controller's command functions, the Ship list in the Model is now a list of Ship_components, and the rest of the code has been updated to support this. This is the easiest way to add Ship_groups to the model while also be able to easily use them in the Controller to tell either Ships or Ship_groups to perform an operation without having to have the Controller know whether it is an individual Ship that they

are controlling or a Ship_group. Virtual functions take care of which function will be called for us.

Another major question is whether a Warship should utilize pointers to Ship_components, or to individual Ships. In our estimation, it would not make sense for a Warship to attack a group of ships all at once. Ship groups are defined by the user, meaning they can be made up of ships from all across the ocean. It would not make sense for a Warship to attack a group since a group has no definitive location to speak of. Therefore, a Warship's target pointer is an individual Ship pointer rather than a pointer to a Ship_component. This can be seen by the Navigability arrow going from the Warship intermediate class to the Ship intermediate class.

The Ship_component class defines a fat interface for all Ship operation functions, including the Ship and the Ship_group class. This interface uses pure-virtual functions since both the Ship and Ship_group classes need to override these functions, so there is not much use defining the functions in the Ship_component class. The Ship class and its derived classes will override these operation functions to tell the individual ships to perform these operations, while the Ship_group class defines these functions to call the same function for all its children.

It is the Ship_group's job to call the set_parent() and reset_parent() functions for all children that it adds to its groups. Whenever an object is added to a group, that group call add_parent() for the child to set the parent pointer (or throw an error if the parent pointer is already set). When removing a child from a group, the parent calls remove_parent(). If a Ship is a member of a group, it is the Ship's job to remove itself from the parent group.

Since a group contains pointers to Ship_components as its children, a group can have other groups as children. In order to avoid a cycle (see the Description document), the add_child() function in the Ship_group class checks that its own group is not a member of child-to-be's group or any of its subgroups. If it is a member of the group or its subgroups, then adding this child would cause a cycle, so the function throws an error.

This extension follows the convention set in Project 4 that says if we store pointers to Ships in another Ship, we should use weak_ptrs in order to prevent cycle dependancies when a ships are being destructed (notably due to the cycle that exists between a child and its parent).

When an operation is called on a Ship_group, we want to pass that operation on to all children, regardless of whether one of the children can't perform the action. Therefore, we use a try-catch loop in all Ship_group operation functions in order to ensure that we are able to pass the operation on to all children, even in the case that one of them throws an error.

Extensibility Statement

Because the class design presented above, any additional ships that are added can be used in a ship group with little to no work. As long as a new ship does not add any new public member functions to the ship hierarchy, then a new ship can be derived from the Ship class and will be able to be a member of a group without having to add any additional code.

If the new ship type requires a new function to be added to the ship hierarchy, then the following should be changed:

1. Modify the Ship_component class to add a pure virtual function as a fat interface for the client to use to call the new function.
2. Modify the Ship class to override the pure virtual function declared in the Ship_component class and throw a descriptive error for classes that do not implement this new functionality.
3. Modify the Ship_group class to override the pure virtual function declared in the Ship_component class, then use the private for_each_child_catch() function to pass the command on to all children in the class.

Beyond the above changes, nothing else needs to be added to the project in order to support adding a new Ship type to the simulation. The new ship type can be added to and used by ship groups with no further work.