- How is the undo/redo design pattern implemented in the model cluster?

I created a deferred class **COMMAND**, which is partially implemented.
It implements the routines that are common to all reversible commands (i.e. moving all projectiles forward or backwards).
Then we have four other classes that inherit from **COMMAND**. These are: **COMMAND_PLAY**, **COMMAND_PASS**, **COMMAND_FIRE**, **COMMAND_MOVE**.
Each of those classes make use of the functionality implemented in **COMMAND** while each is specializing in its own corresponding functionality.

In the main model (**SPACEWAR**), we maintain a list of all valid commands that were executed as well as a cursor pointing at the last command that was executed.
When the user executes **undo**, the cursor reverses the command it's pointing at, then we decrement the cursor, then reverse the command that the cursor is pointing at now, and we re-execute the same command.
When the user executes **redo**, we simply increment the cursor and then execute the command it's pointing at now.
If the user execute a valid command after **undo/redo**, all the commands after the cursor are removed and lost from history.

How are polymorphism and dynamic binding realized in the design at compile/run time?

- Polymorphism
Compile Time: In class **SPACEWAR**, we have a list of commands, the list is an abstract **LIST** where during initialization we assign an **ARRAYED_LIST** to the list variable.
Note: This is how I should've implemented it (the principle of "program from the interface, not the implementation"). But I didn't have time to change things as I have other course work to do.

Run Time: In class **SPACEWAR**, when a command is executed, we add the effective class object of the command (for example **COMMAND_PLAY**) to a list that is declared to contain the objects of type **COMMAND**.

- Dynamic Binding
In the deferred parent class **COMMAND**, we have two deferred routines **perform_command** and **reverse_command**.
How these routines function is dependent on the specific implementation of the effective descendants of **COMMAND** (i.e. the actual commands) since at runtime we look for the specific implementation of the above two routines.