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

Undo/Redo design pattern is implemented in the model cluster by incorporating multi-level undo and redo.

This is done by first having a deferred class COMMAND which will have 2 deferred features, execute and undo. For each different type of user command (i.e move, fire, pass, etc.), a new effective child class under COMMAND will be created.

In these child classes, both execute and undo is implemented. When execute is called, it does the operations for a specific command, e.g MOVE_COMMAND's execute will move the starfighter from one location to another. Execute can also be used to redo; after undoing you can 're-execute' the command back to its new state. Whilst the undo will do the opposite of execute, it'll make it the same as the previous state, e.g MOVE_COMMAND's undo will move the starfighter back to its original (first initialized) location.

To incorporate the multi-level undo and redo we must use a data structure (in our case an array works,) that holds a history of all valid commands that have been called, along with a cursor that points to the index of the current state. In our case, ETF_MODEL will add each error-free command to the history array, each time a new command has been added the cursor will increase by 1.

Alternatively, when undo is called, we first undo the command by locating the current cursor location in the history array. Once located, call .undo to reverse the changes and subtract 1 from the cursor to move it 1 back in history. Suppose we call redo after an undo, it will increase cursor by one and re-execute the command via .execute. For both cases, we must also do error checking to see if we can undo further or redo by checking if the index goes out of bounds, if it does then do not undo/redo. If we were to call a new command, first remove all the commands after the cursor index location in history (i.e from cursor +1 to tail of array) and then add the new command to the history.

How are polymorphism and dynamic binding realized in your design at compile time and runtime?

Polymorphism is realized in my design at compile time by using COMMAND class as my static type. When calling a new user command, it's first called by the feature in ETF_MODEL with the signature of call_command(c: COMMAND). This deferred class is the parent class for the effective user commands (i.e move, fire, pass). When passing a new command to this feature the dynamic type must be of move/fire/pass and it's assigned to variable c (as mentioned earlier in the parameter). In compile time this c is of type COMMAND (static) and then during runtime, it will have the dynamic type of move/fire/pass.

Dynamic binding is realized in my design at compile time by invoking execute and undo routines on the static type of COMMAND. Both routines are deferred and during runtime the dynamic types move/pass/fire will use its own unique execute/undo routine via dynamic binding. Routine execute is executed within call_command in ETF_MODEL (i.e c.execute) and in redo whilst routine undo is executed within ETF_UNDO.
P.S In my actual code, I replaced 'c' with command, was saving space on here!