Lab 3: Design Document

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

The undo is implemented in a way that when the user calls the undo command, the compiler triggers the ETF_UNDO class, in which the UNDO method is called in the classes, “FIRE”, “MOVE”, “PASS”, where the cursor position in the in the array “history” is decremented by 1, and checks if the array contains any of the above 3 classes, it will call its undo method respectively. This undo method in these 3 classes point to the ETF_SD_MAIN_MODEL, where the undo method is called, and the code is successfully executed.

The redo is implemented in a way that when the user calls the redo command, the compiler triggers the ETF_REDO class, in which the EXECUTE method is called in the classes, “FIRE”, “MOVE”, “PASS”, where the cursor position in the array “history” is incremented by 1 , and checks if the array contains any of the above 3 classes, it will call its redo method respectively. This redo method in these 3 classes point to the ETF_SD_MAIN_MODEL, where the redo method is called, and the code is successfully executed.

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

Polymorphism: During compile time, I have history array of type “COMMAND”, in which “FIRE”, “MOVE”, and “PASS” are being assigned into this array as they inherit “COMMAND”. At runtime, each object in the history array will point to different addresses that are descendants of “COMMAND”.

Dynamic Binding: During runtime, “undo” and “execute” methods will defer for each “FIRE”, “MOVE”, and “PASS” COMMANDS, because each command executed will be different. For example, if I give the move command, the execute and redo function of this move will differ from fire or pass commands.