Deadwood Project

CSCI 345 Spring 2019

Gabriel Huffman and Edward Thompson

We did not implement any design patterns directly, but there are a few places in our code that do reflect the basic ideas present in some design patterns. Specifically the observer and factory patterns, and a sort of singleton, were most drawn inspiration from in our code.

The observer is the easiest to see in the code and closest to the original design. Close in that it does draw up information from the object it watches (said object being the board or the current player) and update the board likewise. Our “Observer” is not a concrete thing however, and more of just a function in the controller which asks what it is watching to give it the state that object is currently in. It does not have a way to check if the object has changed or not, merely it is told to update the screen at strategic moments. This I felt is a better way of working with the observer in Deadwood than the traditional one, because when things change in deadwood it tends to be many at once and only at very specific times. Thus there is no need to worry about the updating things one at a time each time something changed and instead update the whole portion of the view which displays the board at specific times in the player’s turn when relevant changes might be happening. This also keeps any slowdown changes in the view would cause to a few moments in the turn. The current player “Observer” does a very similar thing, but drawing information from the player rather than the board itself. This one must be called more times a turn, but is less data intensive thus not causing problems because of that, and still only at specific times.

Our “factories” differ from the design pattern in a very similar way to the observer, namely being a function rather than an object. These factories were very simple, and were used to set up four objects in the game: the jframe, the controller, the game loop, and the board. All of them were called in the controller via the controller building itself, so you might be able to argue they are all one, but it seems more accurate to refer to them as calling each other since each does intensive work to build parts of itself using systematic calls to built in logic or text files containing game information, and those are the four major parts of the game that were really separate from each other, and loosely coupled (except maybe the game loop and board).

There were kind of some singletons being things that only one was built of, but there was no real enforcement on this, merely only one instance of them was called, generally in the controller so it could puppet the view and the model, and held onto the instances so all logic would be using the same one.