One design pattern we found useful in the creation of our prototype was the chain of responsibilities. This design pattern made it easy to pass the results generated by event handlers to the data structures, functions, and classes that would act on them. In particular, the tile matching application was made simpler to implement by using this method. First, an event is recognized when the user clicks on the canvas. Deciding what to do with the large amount of information generated by this click would be overwhelming without slowly narrowing the data to just what is necessary for a matching game. First, we narrow to just the object that was clicked on, and pass that to a class that handles rotating the tile. This object then narrows the information further, by passing only the ids of objects that rotate completely (since the user could have clicked more than two tiles). The next object determines what the colors of the objects clicked were and determines if they match. This is the final step in the chain of responsibilities, and this object/class determines if the user has won.

Additionally, the iterator design pattern was used in both the torus tic-tac-toe and matching game. Iterating over the positions in the double array for tic tac toe was handled by an iterator function checkWin(). A better implementation was in the pattern matching game. The game utilized pre-build iterators for javascript arrays hidden within an object method call for checking for matches within an array as well as for applying the flipping action to multiple tiles at once.