Integration Strategy

For the development of our app, it made the most sense to implement a bottom up integration strategy. This enabled us to separately create javascript games and applications that depended on the canvas 2d context as well as the WebGL renderer. The renderer was accessed using the Three js library, so all 3d applications as well as the matching game used this library. Each of these applications were created in a separate javascript file following the same general format, but implementing their own unique game logic. Each artifact was then able to be tested independently and thoroughly before its final implementation.

In our case, the game written in javascript could be tested completely before being integrated into the web page. Each game was integrated in mainly the same way. We would simply add the link to the src script to the bottom of the html document. Depending on how many games we would end up adding, we would add a script to load only the scripts that the user requested upon selecting a game, but this would not decrease the overall amount of time to download the page, since all of the javascript would need to be loaded anyways.

The advantage to integrating our code in the bottom-up style was that we were able to complete and test games without worrying about how they would show up on the final page. By setting up some boilerplate code that would always append instructions for playing the game and the actual canvas to the same div, we could develop completely separately before integration. After some minor changes to how elements styled with css interacted, the final integration of all of the games into the site went smoothly. Obviously some bugs were only caught after addition to the final page, but these faults were still isolated to the games themselves.