For the final task, we tested the HWMtk’s performance in the Delayed Saccade task. We chose this task because it was the main task that was used in the past to demonstrate the capability of the original WMtk. Because of this, we already had the DST source code, but more importantly, we were able to test both toolkits on the same task. This provided us with a strong comparison of performance between the original and augmented toolkits. We created a version of the DST that was wired to the HWMtk, replacing the DE-encoding functions in the old interface with simple function calls to the HWMtk, passing in a simple string representation for each state. We then run the DST 100 times on both toolkits, recording the number of episodes taken to learn the task to within a 98% success rate within a window of 20 episodes.

Setting up the DST using the HWMtk proved simple compared to setting it up using the original toolkit. When the DST was written using the WMtk, the first parts of the task that had to be written were the DE/SE conversion functions that took the data structures describing the chunks of information and the state being passed to the toolkit and converted them into representations that the WMtk could use. An example of this kind of distributed coding is shown in figure 13. Using the HWMtk, we merely had to pass a string containing that information to the WM object, which then handed it off to the HRRE for encoding into HRRs. We were able to immediately jump into writing the logic for the task without worrying about the overhead of manually writing DE conversion functions. Rather than write a reward function to hand to the toolkit, we were also able to write the performance logic directly into the task driver code, simply running a series of checks to see if the agent has been taking the proper actions, and calling Absorb Reward with the appropriate reward value upon completion of the episode.