Stephen Cardone  
Operating Platforms CS-230-J3533  
Professor Watson

Module 6 – Storage and Memory Management

Memory management can be an important ting to consider when developing software. In the case of Draw It or Lose It, it should not be too difficult to set up a successful memory management paradigm. For Draw It or Lose It, we will need to store around 200 individual 8MB images. This equates to around 1.6 GB of storage space. We will also need room to store user data. Currently there is not much data associated with each user, but we will need to allocate space for an increase in user count. We should allocate at least 10GB to allow for the expansion of the user base. With these considerations, we would expect to have a server sided storage of about 16GB or more. In reality, this is tiny, so we should plan to use 256-512GB to account for expansion. To further detail the storage and memory for Draw It or Lose It, we will need to explore the client side memory and storage, by assessing device capability, and user experience.

On the client side, we want the users to have a seamless and rapid image experience. One way to achieve this seamless experience would be to have users store or cache the images locally on their device. This would work well because users would not need to wait for an image to download each time an image is displayed in the game. This would also benefit the users by using up less bandwidth, by allowing the users to install the 8MB images once, instead of over and over again is inefficient. If we update the game to add new images, or the user clears their cache or deletes the Draw It Or Lose It storage, then they will download the images again at that time. Once the images are stored on the device or cached, we will read the images into memory to be used by the app then cycle the image out of memory to make room for the next images. If the memory allows for it, we could read the next image into memory while the previous is still displayed. This would allow us to buffer our image before we need it, allowing us to show it on the screen quickly in low amounts of time. The downside to his strategy is that we would need to have multiple image in memory at once which is a burden on memory.

One final issue to be considered is how the images will be slowly displayed to the screen. The game is designed to slowly show the images to the screen. We would want to avoid having many copies of the same image that are in different stages of the drawing cycle. It would be better to take the image an process it by clipping on the device side, or we could overlay opaque screen areas that slowly disappear to reveal more and more of the image over time. Having many partial images for a given image should be avoided at all costs.