**Memory Management**

When developing “Draw It or Lose It” to be a cross-platform application that serves many clients, some considerations into the management of memory will need to be made.

Memory affects the speed at which our server can process information and send it back to a client. Since there will be several different platforms on which the application will be running, performance may vary on the client-side. We will want the server-side to be high-performing to ensure a consistent experience as possible, no matter the platform.

We should also plan for best-case and worst-case scenario when it comes to number of users. The more users we have, the more memory we will need. If an unexpected number of users enter the game, we should have the resources ready for that to avoid a crash. We can, in a pinch, use paging/swap memory where needed, although dedicated memory would be preferable (Performance Considerations, 2018).

Another consideration would be garbage collection in our application. For the most part, this can be taken care of automatically by our chosen programming language. When an object is no longer needed, it is destroyed and its allocated memory returns to the heap, available for the program to use again. We should try and be aware of events that force garbage collection or make them last longer than normal. As an example, if multiple objects are created in the innermost part of a for-loop that runs multiple times, a garbage collection event would occur for each of those objects and possibly degrade performance of our app (Overview Memory, 2020).

**Storage Management**

Considerations will also need to be made regarding the storage required for our application. We need to store images for our game, and the profiles of users that are playing.

Our application uses 200 high definition images that are approximately 8 megabytes in size. All in all, this is not very much, but we may want to have redundancy built in to protect against crashes, data loss, or outages.

We will also need a way to store user profiles. An accepted way to handle this is via a database. Using our chosen server architecture, we can make calls to our database which will return information about our user that we can display to the clients.

Since our application uses libraries, something to consider is that we will need to include those libraries in the storage of our server. This can cost extra if we go with a serverless method for our application as we will typically pay per unit of storage. On a related note, when storing our application in its area of deployment, the file hierarchy is an important part of its functionality. This is important to observe because file hierarchy is an added cost on serverless architectures.

**Comparison**

As discussed, memory and storage are two different computing concepts with different implications for our application. Memory, broadly, will affect the performance of our application. If we have more memory, our server will be faster to compute data and communicate with each client. Storage on the other hand is where we can keep assets that our application will need and can come in different forms such as hierarchal, block, or database. The more storage we have, the more assets we can have.

It is also important to consider the actual utilization of memory and storage by our application. We don’t want so little that our game fails to function properly and provides a poor experience for our users, but we also don’t want so much that we have a lot of unused resources in which we have wasted money on. But we would also like to scale in the future. There are many scenarios to think about.

Today, we generally have two options when it comes to serving an application. We can make it serverless or traditional. A serverless application will use pay-per-use microservices on a platform managed by a third party, such as Amazon Web Services. This gives us the ability to work solely on our application and not have to worry about maintaining a server. A traditional server approach would mean that we need to purchase and maintain a server capable of serving many clients but gives us control over our own systems (Arsov, 2019). This knowledge is important when considering the memory and storage implications of our application because either method of serving will affect the overall budget for the program.

**Bibliography:**

Server Hardware Performance Considerations. (2018, January 8). Retrieved August 10, 2020, from https://docs.microsoft.com/en-us/windows-server/administration/performance-tuning/hardware/

Overview of memory management :   Android Developers. (2020, May 6). Retrieved August 09, 2020, from <https://developer.android.com/topic/performance/memory-overview>

Arsov, K. (2019, July 01). Microservices vs. SOA - Is There Any Difference at All? Retrieved August 10, 2020, from https://medium.com/@kikchee/microservices-vs-soa-is-there-any-difference-at-all-2a1e3b66e1be