# Garbage collection

After running for a long time with a simple non-collecting memory manager we finally managed to get a mark-sweep implementation working. The layout of objects basically stayed the same as it was already designed to support GC. The thing that did change is that recorded object size can be larger than requested when allocating. This is due to the way we track holes in the heap. Each hole records its size in the first word and a pointer to the next hole in its last word. Because the common object header takes two words any useful objects takes up at least three words and so a hole of length less than three is useless – at least until one of the objects around it is collected. Instead of waiting for the death of a surrounding object any time a whole of size less than three is created we simply treat it as though the allocation filled the entire hole. The reference from the previous hole is thus updated to jump over the eliminated hole. However because the recorded size of an object is used to scan the heap during sweeping we need to pad the object just allocated with the size of the hole making it one or two words longer. Not realizing this from the beginning caused a lot of problems.

The root set is quite easy for us to obtain. Firstly we take all of our handles which are recorded centrally along with any references stored on the stacks of any running interpreters. Since we use two words per stack element (class and pointer) sorting integers from objects is quite easy. To make updating of handles and stack elements easy when moving objects a dictionary is created mapping each root to a delegate making the appropriate update. Once this dictionary has been created it is given to the marking phase. Running through each root the marker extracts the object type and calls a method that return all the references in that object ensuring that all reachable objects are marked. The allocation, marking and sweeping implementations themselves algorithm itself are straight forward.

While we did not have time to implement compaction (making our ability to actually tracking object moves pointless) we did manage to implement heap expansion. Currently, if an allocation fails, a garbage collection is initiated and the allocation is retried. If it still fails the heap is expanded and we retry again. In the end if the maximum specified heap size is reached and the allocation still cannot be made an out of memory exception is thrown.