Assessment Task 1 – Design Game Optimisations Design Doc

Object pooling

I’ve decided to implement a object pool into the code for two big reasons 1. Being it was mention in the code with a comment twice “// its pretty inefficient to keep reloading textures. ...if only there was something else we could do “and “// this would be the perfect time to put the critter into an object pool “

The main advantage of adding an object pool is it stops memory fragmentation and speeds up performance every time we need to create/respawn a critter.

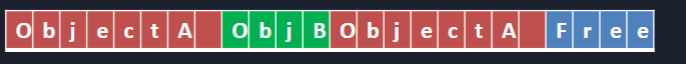
So at initialization we allocate memory for all the critters [50] and add a bool variable to the critter class to check if the critter is active/alive and we add a check to the draw function where if it is not alive we simply don’t draw it while still keeping it in memory

This helps with performance since with the original code we were unloading the texture and then re initializing the object at run time which massively slows down performance with the approach I just mentioned it can also because something called memory fragmentation.

This is when you have objects completely filling up a chunk of memory and when you deallocate some of those objects to free up space, but if you don’t rearrange the memory those small gaps might be completely useless, there could be enough room for a large object but since the memory is fragmented it means you would have to fragment the large object as well which we cant do, so this will waste much more memory then needed  
  
so to get around this we keep those small objects in the memory without deallocating them and just keep track of them being activated or deactivated.

And I’m stealing a visual representation from the object pooling slides from AIE to better demonstrate this concept.



Here we have a chunk of memory.  
  


We allocate our objects to memory.



We free up memory by deallocating Obj B.



Now here is the memory fragmentation we have enough free memory for another Object A but since the memory is fragmented, we will have to use up another chunk of memory which is wasting resources.

Then with a draw function we just have a simple function to check if its active or not   
  
For (critters[i]<MAXCRITTERCOUNT){

If(critter[i]is active){critter[i].Draw;}  
  
}

Resource management

Resource management helps save on memory using the assessment as an example when we initialize the 50 critters, we load 50 copies of the exact same texture which takes up a lot of memory and slows down performance for each critter we have on screen, and this problem scales with how many critters we have on screen, so how do we go about optimising.

When we Initialize the 50 critters instead of each critter loading in its copy of the texture, we use the resource manager to load one copy of the texture then hand out a reference of that texture to every critter this saves on memory which will improve performance.

//insert class diagram here lmao!!!

//and maybe add a diagram for showing how it works