Optimisation Design Brief

Texture Loading:A computer screen with text and images

Description automatically generated

A computer screen with text

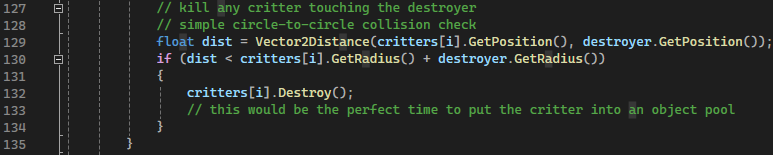
Description automatically generated

A screen shot of a computer code

Description automatically generated

The texture loading needs optimisation because as the program is now, whenever a critter gets made it loads a texture, and when one gets destroyed it unloads a texture. Because of how often this happens, the program is slowed down.

To fix this, I implemented a hash table that takes in Texture2Ds, which stores the already loaded textures for the critters and the destroyer. I chose to do this as a hash table seemed like the most efficient way to store the textures. From there, I changed the function for making critters so that rather than taking in a character array, which would have been the name of the texture file, it takes in the loaded Texture2Ds from the hash table. I chose to do this because it means that now instead of loading textures every time a new critter is made, textures are only loaded once at the start of the program. Finally, I changed the critter’s destroy function so that it no longer unloads textures, and instead doesn’t unload at all. I chose to do this as the textures already get unloaded when the window is closed.

Object Pool:

This part of the code needs optimising, as without an object pool, there is a risk of memory fragmentation, where space that should be available to store a critter is unavailable. Adding an object pool would allow the program to take critters from a list of critters, instead of frequently making new ones.

To do this I made a new class for object pools, which acts as an array of critters. I did this so that the critters can be stored inside of it. From there, I made a function for allocating and deallocating, so that critters can be added to the object pool.