Object pooling

What is being improved

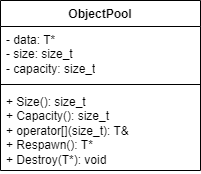
Currently the critters in the project are checking collision against dead critters or are implementing a check to determine if the critter in question is dead before continuing with the function. To increase the performance of the solution, it would be best to implement object pooling for dead critters to reduce the number of checks that need to be made and completely remove the need to check if a critter is dead.

How it will be implemented

To implement these changes a class should be made to handle sorting dead critters to the end of the array and manipulating the dead critter starting placement as they are removed and added. This placement can be used in place of the current isdead checks and critter count in the loops.

Reason for implementing it as described

Object pooling was a pretty clear choice for an optimisation considering it can be implemented and affect instances throughout the code and removes the need for a lot of unnecessary checks that are in place while streamlining the respawning of dead critters significantly by not having to loop through all the alive critters to find one dead critter to respawn.



Improve collision detection

What is being improved

The solution makes redundant and expensive collision checks by comparing all the critters against each other with frequent redundant checks between critters. It also compares the square root of the distance which is a heavy operation to perform multiple times every frame. You can simplify both of these by skipping over critters that have already been tested for collision and getting the square values of both distance and radius and using those rather than obtaining the root of one.

How it will be implemented

In the current implementation the critter to critter collision checks iterate through each critter one by one then every other critter one by one, the first critter is referred to as i while the second is referred to as j. This check can be simplified by making j start at the value of i so that it doesn’t do redundant collision checks between critters that have already been covered and hence changing a O(n^2)-n to a (n-1)\*n/2 time complexity.

The conversion from using square root comparisons to squared comparisons is a simple implementation that only involves creating a new function for obtaining the squared magnitude (as the current magnitude calculation gets the square root magnitude) and comparing it against the squared radius of the critters.

Reason for implementing it as described

These two solutions were chosen because they cleaned up the project and reduced the computational time drastically by removing the heaviest operation in the program (finding the square root of the squared magnitude), and significantly improved the time complexity of the collision by around half. They were also the prominent hot spots when using the performance profiler in visual studio.