# 3D AGENT BASED AQUATIC SYSTEM

**Approach:**

Arrangement of the fishes: Three species of fishes floating in the aquatic system to be arranged are:

* minnows
* tuna
* sharks

std::vector is preferable for fast access, easy maintenance, min overhead. But, space consumed = O (N^3) space, where N is the number of grid points along one dimension. So, here I have used std::map. The usage of key for every data point gives fast access to the data associated. This gives me a combination of speed and space O(N).

Map Structure:

Key fish

my\_map

Step Num

Num fishes

Ate or not

**Files included:**

* main.cpp
* functions.h & functions.h: for the aquatic system
* mapCoordinates.h & mapCoordinates.cpp: for setting the directions of the map
* myrand.h & myrand.cpp: for random values’ generation, uses “random” library from C++11

Additional updates: usage of various classes, class inheritance, typedef declaration, C++11 library random, auto used; assert used, try-throw-catch, to get if all the fishes are alive (to continue the system or not)

**Conclusions:**

For determine initial conditions for (Nshark, Ntuna, Nminnow) such that each of the three species has a non-zero population after 1000 sweeps.

Considering, (Iteration =1000) + (Sweeps = 125), all the fishes show alive at any num of initial count. Most of the times, one of the fish is dying at ~57th iteration (=> 57x125 steps = ~7k steps)

**Pseudo code:**

|  |  |  |  |
| --- | --- | --- | --- |
| create the system | | Create the map | Use Typedef |
| initiate the system | | random alloc of fishes at sites | void initMap() |
| select fish randomly |  | Select one of the three fishes from the system (selects non-empty location) | char chooseFish() |
| For 1000 sweeps => for (auto 1: 1000) //slide 197 | | | |
| For each sweep => for (auto 1:125) //sweep size L^3 | | | |
| Check fish status | If Tuna/Shark | Kill the fishes that didn’t eat in last 5 trials, provided trials > 5 | Void killFish(); |
| Move fish |  | Gets the site to update after checking the below functions and inserts the fish in the site | Int moveFish (); |
| If Minnow | All directions have same probab=>one step any axis |
| If Tuna | Same axis->diagonal=>one step same axis + one step different axis |
| If Shark | Same axis->2steps one axis, 1 step |
| Update site status |  | Based on the arrival site status & attacking site status, | Void updateArrivalSite() |
| 2Minnow | +3Minnow |
| 2Tuna + ate | +Tuna |
| 2Shark + ate | +Shark |
| >Tuna + Minn | -Minnows |
| >1Tuna+>1Shark | -1Tuna |
| >1Shark + Minns | -Minnows, in neighboring sites too based on feeding frenzy conditions |