Massing | Living Unit Agent

```
//The algorithm below is one of the agents used during the growth of the massing.
//It is based on a 2-way optimized weighted growth, with each function trying to get
//into an equilibrium between optimal conditions and relations between each other.

//The input is a point cloud with various site specific attributes
//and defined unique starting points for each function.
//First, it looks at the current point's distance to the function centre and depending on it deletes it.
//Secondly while it still needs to grow, it looks at all the neighbouring points
//and stores the id of the best one.
//Otherwise, if it has reached its maximum it keeps checking its worst point against potential new ones
//and deletes chooses the better until it has reached the optimum
//Lastly, it assigns the found point to the given function

//The output is a point cloud with the given function having its assigned points

int group [] = expandpointgroup(0, "Living_Unit");
int runs = len(group);

if (@Frame % 50 == 0){ //Let it search for new voxels every fifty frames
    setdetailattrib(0, "liv_growth", 0, "set");
}

float growth = @liv_growth;
float v_dist = chf("../../../../../Voxels/Voxel_Size/v_height");
float v_side = chf("../../../../../Voxels/Voxel_Size/v_length");
// Living Unit
int liv_id = 5000000000000000000000;
float liv_perf = 0;
float temp_liv_perf = 0;

//Agent Attributes
int liv_floors = chi("../../../../Living/floor");
int liv_area = chi("../../../../Living/area");
int req_vox = int(rint((liv_area/(pow(v_side,2))) * (3/v_dist)));
int vox_amo = npointsgroup(0,"Living_Unit");

float liv_sun = chf("../../../../Living/sun_perc");
int liv_min_floor = chi("../../../../Living/min_floor");
int liv_max_floor = chi("../../../../Living/max_floor");
int liv_qt = chi("../../../../Living/r_noise");
float liv_con = chf("../../../../Living/con");

if (growth == 0){
    for (int i = 0; i < runs; i++){
        //Calculate the function centre
        vector curr = point(0, "P", group[i]);
        int pgloc [] = expandpointgroup(0, "Living_Unit");
        int pgloc_len = len(pgloc);
        vector loc = {0,0,0};
        for(int k = 0; k < pgloc_len; k++){
            vector temp = pointattrib(0,"P",pgloc[k],1);
            loc = loc + temp;
```

```
    }

    loc /= pgloc_len;
    setdetailattrib(0, "Living_Centre", loc, "set");

    if(max(abs(curr.x - loc.x), abs(curr.z - loc.z)) > sqrt(liv_area/liv_floors) * 0.8 || abs(curr.y - loc.y) > 3 * liv_floors / 2){ //Check if is not too far from the function Centre
        setpointattrib(0,'func_id', group[i], 0 ,"set");
        setpointgroup(0, "Living_Unit", group[i], 0, "set");
    }
    else{
        // Getting Point Attributes
        int func_id = pointattrib(0, "func_id", group[i], 1);
        vector base = point(0, "P", group[i]);
        int liv [] = nearpoints(0, base , v_dist);

        for( int j = 0; j<len(liv); j++){
            int point_func_id = pointattrib(0, "func_id", liv[j], 1);
            float sun_rate = pointattrib(0, "Sun_Rate", liv[j], 1);
            int floor = pointattrib(0, "Floor", liv[j], 1);
            float dB = pointattrib(0, "dB", liv[j], 1);
            float dB_norm = pointattrib(0, "dB_norm", liv[j], 1);
            float con = pointattrib(0, "Connectivity", liv[j], 1);
            vector pt = pointattrib(0, "P", liv[j], 1);
            int occup = pointattrib(0, "liv_occupied", liv[j], 1);

            //Living Unit Performance
            if (floor >= liv_min_floor && floor <= liv_max_floor && point_func_id == 0 && con >= liv_con){ // Checking if voxel is on the right floor, connectivity and is free
                if (sun_rate >= liv_sun && rint(dB) <= liv_qt ){ // Checking for sun and noise
                    int ng [] = nearpoints(0, "Living_Unit", pt, v_dist);
                    temp_liv_perf = pointattrib(0, "living_perf", liv[j], 1);

                    if (temp_liv_perf > liv_perf && occup < 1 && vox_amo < req_vox){
                        liv_id = liv[j];
                        liv_perf = temp_liv_perf;
                    }
                }
            }
        }
    }
}

if(vox_amo >= req_vox && growth == 0){ // If it is full but still wants to grow
    int gp_pt [] = expandpointgroup(0, "Living_Unit");
    int gp_run = len(gp_pt);
    float pt_dist [];
    vector centre = @Living_Centre;

    for(int m = 0; m < gp_run; m++){
        vector ref = pointattrib(0, "P", gp_pt[m], 1);
        float tem = rint(distance(centre, ref)*100)/100;

        pt_dist[m] = tem;
    }

```

```
        float max = max(pt_dist);
        int ind = find(pt_dist,max);

        float comp_liv_perf = pointattrib(0, "living_perf", gp_pt[ind], 1);

        //Compare the performance of the furthest from its function centre and best it could grow to
        //Choose to better one, if there are no better ones stop growing
        if (comp_liv_perf < temp_liv_perf){
            setpointattrib(0,'func_id', gp_pt[ind], 0 ,"set");
            setpointgroup(0, "Living_Unit", gp_pt[ind], 0, "set");
        }

        else{
            setdetailattrib(0, "liv_growth", 1, "set");
        }
    }
    // Assigning the functions
    if (liv_id != 5000000000000000000000000){
        setpointattrib(0,'liv_occupied', liv_id, 1 ,"add");
        setpointattrib(0,'func_id', liv_id, 1 ,"set");
        setpointgroup(0, "Living_Unit", liv_id, 1, "set");
    }
}
```