# Algorithme génétique: apprendre à voler (2D).

Objectif :

Créer des objets qui se déplacent dans un espace limité et qui ne se touchent pas ni ne touchent aux limites du volume.

## Besoin :

Pour chaque objet, paramètres :

Position x, y, z,

Vitesse Vx, Vy, Vz,

Distance d’un objet adjacent (dx, dy,dz) (calculé par fontion : CalculDistance())

(Vitesse de rapprochement de l’objet adjacent : Vrx, Vry, Vrz.

Matrice de paramètres (neural network)

Temps, distance (ou temps) parcourue avant de frapper un obstacle (fitness evaluation).

Fonction d’ajustement de la vitesse en fonction de l’output du NN (dVi)

Fonction de dessin de l’objet?

Fonction pour sauvegarder les paramètres de la matrice.

Fonction pour charger les paramètres de la matrice.

Fonction pour mutation de paramètres.

Fonction pour combiner les paramètres de deux ou plusieurs objets (reproduction).

Algorithme de calcul de distances (multithreaded)

Le résultat du calcul est utilisé pour influencer la direction de l’objet. La distance la plus courte est un paramètre d’entrée du réseau neuronal (dx,dy,dz).

Réinitialiser toutes les distances. Possibilité : distances = distance entre objet et les limites du volume (valeur initiale).

Fonction : CalculDistanceInterObjet()

For (objetSource =0, objetSource < numObjet)

For (objetDestination =(objetsource +1), objetDestination< numObjet)

Calcule distance (objetSource, objetDestination)

Pour chaque objet (source, destination)

Si la distance calculée est plus courte que distance initiale (dx, dy,dz)

Alors {mettre à jour la distance min de l’objet}

(numObjet = nombre d’objet en évaluation)

Fonction : CalculDistanceVolume()

For(objetSource=0, objetSource < numObjet)

Calcule distance Objet -Mur(1..4)

Pour distance 0-x: la position « x » est la distance.

Pour la distance x-MurX : la distance est PositionMurX- « x »

Répéter pour Y et Z.

1. Besoin de déterminer quelles sont les plus courtes distances, les coordonnées et ID de l’objet situé à la distance calculée.
2. Si la distance est trop courte (collision), arrêter le mouvement de l’objet, marquer son unité de fitness (distance parcourue, temps écoulé, etc).

Changement de vitesse :

La vitesse est constante, donc V2 = Vx2 + Vy2 + Vz2.

Lorsqu’un changement de vitesse a lieu :

V2 = (Vx + dVx)2 + (Vy + dVy)2 +(Vz + dVz)2, seulement une variation dans une direction est effectuée.

Le changement de vitesse est inverse à la vitesse relative des objets les plus près. Ex : si l’objet le plus proche se rapproche (Vxd -Vxs < 0), alors la vitesse dans cette direction doit diminuer.

Si la vitesse de l’objet le plus près est plus grande que la source (Vxd – Vxs > 0), alors la vitesse reste constante ou peut diminuer pour augmenter la distance.

Vitesse relative :

V(relative)=(Vd-Vs)\*sign(Xd-Xs)

Si Vr <0, les objets se rapprochent, si Vr >0, ils s’éloignent.

Réseau neuronal :

Avec les paramètres d’entrée, la sortie est la variation de vitesse à appliquer à l’objet afin d’éviter : dVx, dVy, dVz

Inputs : distance de l’objet le plus proche : dx, dy, dz, vitesses relative (approche, éloigne) : VRx, VRy, VRz.

Outputs : changement de vitesse (dVx, dVy, dVz). Ces valeurs servent à ajuster les paramètres de vitesse de l’objet (Vx, Vy, Vz). La vitesse doit demeurer constante dans cette version initiale.

Multithread :

#include <cstdlib>

#include <iostream>

#include <string>

#include <vector>

#include <ctime>

#include <numeric>

#include <cmath>

#include <sstream>

#include <thread>

#include <chrono>

#include <ctime>

#include <mutex>

int GetRandom(int max) {

srand(time(NULL));

return rand() % max;

}

// ---------- CALCULATE PRIMES WITH THREADS -----------

// Used to protect writing to the vector

std::mutex vectLock;

std::vector<unsigned int> primeVect;

void FindPrimes(unsigned int start,

unsigned int end) {

// Cycle through numbers while ignoring evens

for (unsigned int x = start; x <= end; x += 2) {

// If a modulus is 0 we know it isn't prime

for (unsigned int y = 2; y < x; y++) {

if ((x % y) == 0) {

break;

}

else if ((y + 1) == x) {

vectLock.lock();

primeVect.push\_back(x);

vectLock.unlock();

}

}

}

}

void FindPrimesWithThreads(unsigned int start,

unsigned int end,

unsigned int numThreads) {

std::vector<std::thread> threadVect;

// Divide up the calculation so each thread

// operates on different primes

unsigned int threadSpread = end / numThreads;

unsigned int newEnd = start + threadSpread - 1;

// Create prime list for each thread

for (unsigned int x = 0; x < numThreads; x++) {

threadVect.emplace\_back(FindPrimes,

start, newEnd);

start += threadSpread;

newEnd += threadSpread;

}

for (auto& t : threadVect) {

t.join();

}

}

int main()

{

// Get time before code starts executing

int startTime = clock();

FindPrimesWithThreads(1, 200000,7);

// Get time after execution

int endTime = clock();

// Print out the number of seconds

std::cout << "Execution Time : " <<

(endTime - startTime) / double(CLOCKS\_PER\_SEC)

<< std::endl;

// for (auto i : primeVect)

// std::cout << i << "\n";

return 0;

}

// ---------- END CALCULATE PRIMES WITH THREADS -----------

// ---------- END C++ TUTORIAL 16 ----------