RANDRIANASOLO Herinina Jenny Stephan
N° 15607284
Machine parallèle

Projet implementation problème du voyageur avec l'algorithme des fourmis

Compilation : make
Usage : ./antcoloni [NBR_ROAD] [NBR_ANT]

Le principe de ce programme, c'est que tant qu'une fourmi n'a pas visité tous les noeuds, il se déplacera vers les meilleurs chemins (meilleur pheromone et courte distance) qu'il n'a pas encore visités.

Dans le programme je considère qu'une fourmi et un thread et se partageront les données du graphe. Ici j'utilise un std: : shared_mutex pour la parallelisation du programme. Dans la fonction ant Move()

```
/*
* param & graphe : le graphe
* param & Ant : le fourmi
*
* Dans cette fonction, tant que le fourmi
* n'a pas visiter tous les noeud, il se deplacera
* vers les meilleurs chemin qu'il n'a pas encore visiter
* et qui aura le plus de phéromone.
* retrun Node
*/
void antMove(Node & graphe, Ant & ant, int nbr_road){
```

Quand les fourmis consultent les données du graphe j'utilise un shared lock sur les mutes car ici on ne fait que consulter les données.

```
// check best pheromone among child road
{
    std::shared_lock lock(mutex );
    for(auto & road: graphe.childs.at(ant.current_pos_id).childs){
        if((int) ant.road_visited.size() < nbr_road){
            if(!isInAntVisited(road.first ,ant.road_visited)) {
                float pheromone = (road.second.pheromone / road.second.nbr_visited) / road.second.distance_from_origine;
                if(best_pheromone < pheromone){
                    best_pheromone = pheromone;
                    best_road_id = getRoadIdByName(graphe, road.second.road_name);
                    best_road = road;
                }
            }
        }
        else{
            if(road.first == ant.road_visited.front()) {
                float pheromone = (road.second.pheromone / road.second.nbr_visited) / road.second.distance_from_origine;
                best_pheromone = pheromone;
                best_road_id = getRoadIdByName(graphe, road.second.road_name);
                best_road = road;
            }
        }

    }
}
```

Et lors des mises à jour des pheromones des chemins dans le graphe, j'utilise un unique lock pour sécuriser les données.

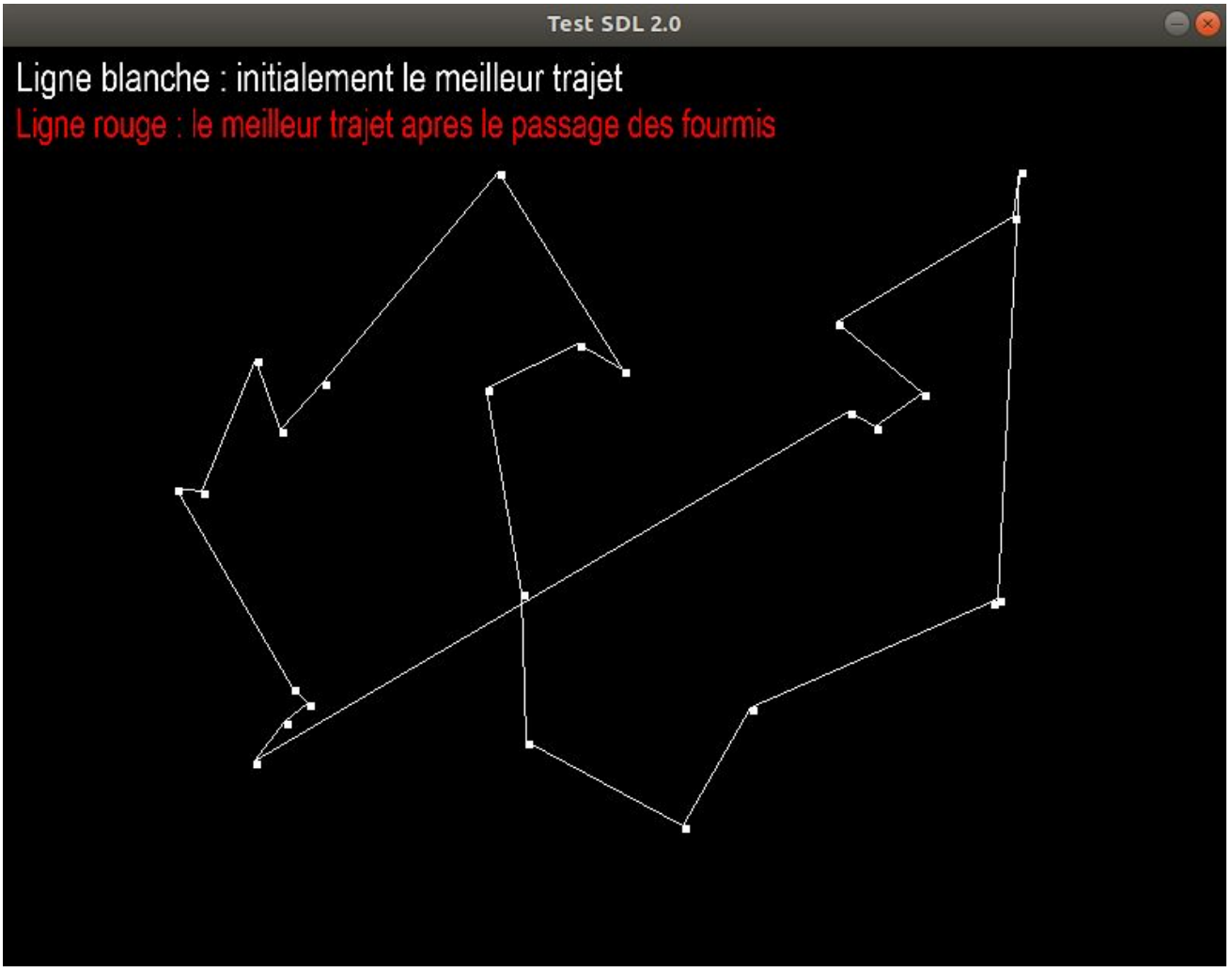
```
//check accesibility and update data
{
    if(best_road_id >= 0 && best_pheromone >= 0.0){
        std::unique_lock lock(mutex_);

        new_pheromone = (best_road.second.pheromone/2) + ((1/(nbr_road*best_pheromone)) * best_road.second.nbr_visited);
        for(auto & road: graphe.childs.at(ant.current_pos_id).childs){
            if(road.second.road_name.compare(best_road.second.road_name)){
                road.second.pheromone = new_pheromone;
                break;
            }
        }

        for(auto & road: graphe.childs.at(best_road_id).childs){
            if(road.second.road_name.compare(graphe.childs.at(ant.current_pos_id).road_name)){
                road.second.pheromone = new_pheromone;
                break;
            }
        }

        graphe.childs.at(ant.current_pos_id).childs.at(best_road_id).nbr_visited++;
        ant.current_pos_id = best_road_id;
        ant.road_visited.push_back(best_road_id);
    }
}
```

Demo du programme :
Initialement le meilleurs chemin



Le meilleurs cemin apres le passage des fourmis