ARCHADE Archimède
KABA Lansine
LUTZ Nicolas
SCHEIDEL Arthur

UNIVERSITÉ DE STRASBOURG

Projet Systèmes Distribués
L3S6 - 2015

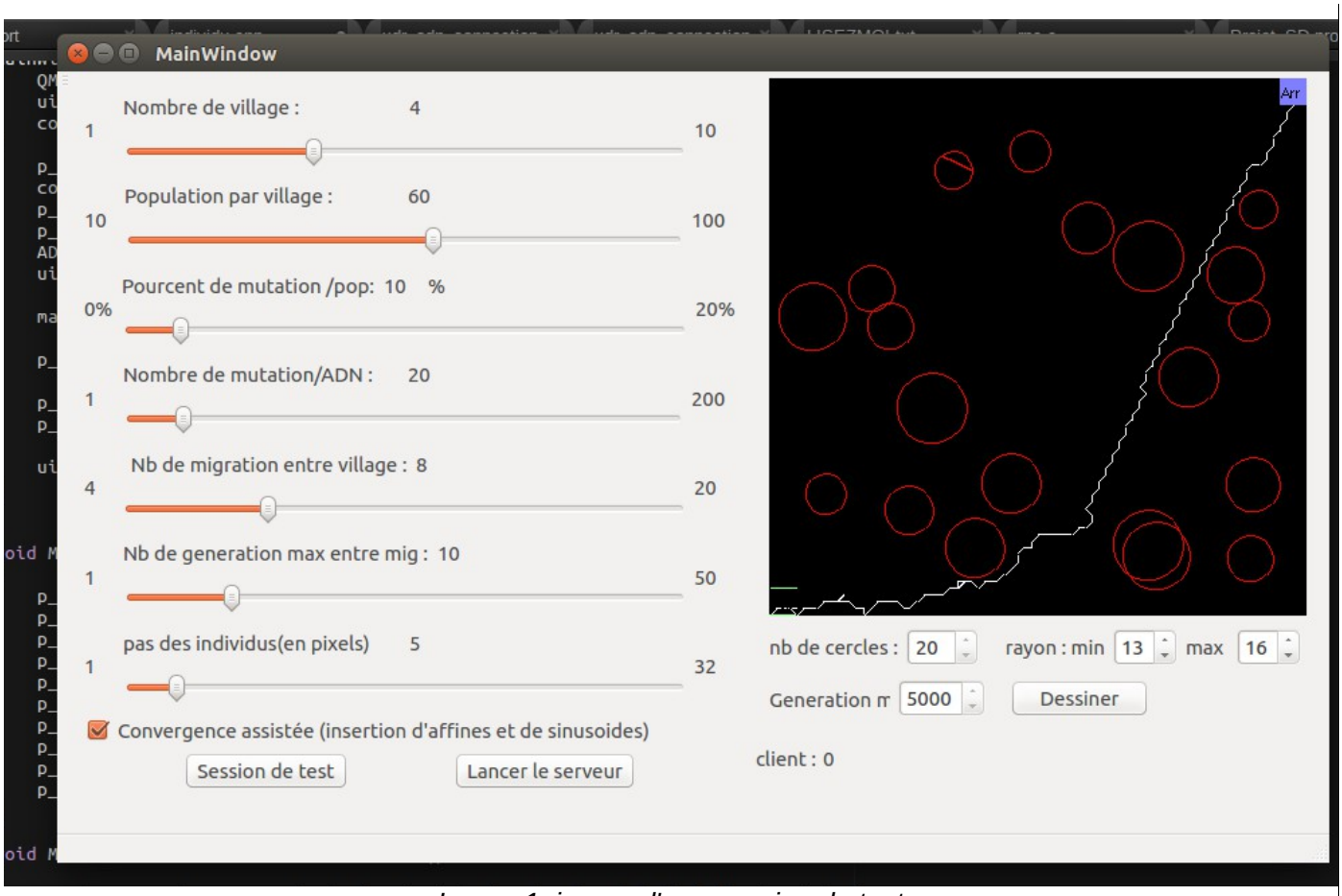

Image 1: image d'une session de test

Responsable TP et Projet :
SONNTAG Benoît

Introduction

Le but du projet est de réaliser un algorithme génétique calculant le meilleur chemin d'un point de départ à une arrivée à l'aide **d'individus** possédant un ADN leur donnant leurs déplacements ; organisés en **villages** qui sont distribué sur des **clients** ; un **serveur** s'occupe de distribuer la **map** et les paramètres initiaux puis un serveur d'échange récupère le meilleurs chemin pour l'afficher à chaque nouvelle génération et échange certains individus entre les villages de temps en temps.

Choix de modélisation

Nous avons écrit le projet en langage C++ en utilisant la bibliothèque graphique Qt, la bibliothèque 3D OpenGL et la technologie RPC pour la distribution des calculs sous forme de client/serveur. Nous organisons notre projet en deux fenêtres : **une fenêtre principale** qui permet de choisir tous les paramètres variables avant le lancement d'un serveur, avec une **fenêtre OpenGl** imbriquée affichant la map et le meilleur chemin actuel ; **une fenêtre de console** démarrée à côté, en même temps que la fenêtre principale, qui servira à afficher des informations complémentaires.

La map et le meilleur chemin final sont conservés à la fin d'une session pour en refaire une directement sans quitter le programme.

Modélisation de l'univers

La fenêtre OpenGL 2D est de 400x400pixels. Un déplacement est représenté par une ligne, d'un nombre de pixel à définir, allant dans une des 8 directions possibles. Les obstacles sont des cercles au nombre de 5 à 20, de taille variables allant de 5px a 20px de rayon. Ils sont générés aléatoirement et il est possible de les redessiner pour tester un autre parcours.

Algorithme génétique

Le meilleur chemin est obtenu par brassage de la population sur de nombreuses générations suivant un algorithme génétique. Pour ce faire, nous imitons le comportement naturel de reproduction, de mutation et de migration sur les individus de chaque village.

Déplacement et collisions

L’ADN d’un individu représente ses déplacements donc on peut calculer une position relative à la fenêtre pour savoir s’il y a une collision avec un des bords de la carte ou avec des obstacles (test si la distance de l’individu aux centres des obstacles est inférieure à leurs rayons). On retient le nombre de pas et la dernière position d’un individu avant une collision ou à la fin de son ADN pour calculer sa distance de la sortie qui nous servira pour son évaluation.

Classes et organisation générale du code

Nous avons programmé sur QtCreator qui utilise un créateur graphique de fenêtre (QtDesigner) qui en génère automatiquement le code (technique utilisé pour dessiner la fenêtre de la console et celle de la fenêtre principale). Il ne s'agit que d'un descripteur physique de l'apparence des fenêtres, et ne gère en rien leur comportement.

Nous avons donc un code découpé en 3 modules (l'interface homme-machine, l'algorithme génétique et le code XDR/RPC)

IHM

Elle initialise tous les autres modules en demandant à l'utilisateur de renseigner tous les paramètres nécessaires au lancement d'une session.

Le module contient les classes *MainWindow* (fenêtre de QtDesigner), *glWidget* pour la map OpenGL et *Vector* pour les dessins. *MainWindow* permet de choisir :

> *Le nombre de villages ; le nombre d'individus par village ; le taux maximal de mutation parmi les individus ; le nombre d'éléments maximaux a muté dans un ADN ; le nombre d'individus migrants par village (proportionnel au nombre de village pour assurer une distribution équitable) ; le nombre de générations maximales entre deux migrations (puisqu'elles n'ont pas lieu à chaque fois) ; le pas d'un déplacement (en pixel) ; le nombre de cercles ; le rayon minimal et maximal des cercles ; le nombre de générations maximal (pour accélérer les tests) ; la possibilité d'assister les ADN par des affines et des sinusoïdes...*

On peut démarrer une session de test sans distribution (et donc avec un seul village) ou bien démarrer le serveur qui attendra qu'autant de clients qu'on a paramétré de villages se soit connectés pour ensuite démarrer la session automatiquement.

La fenêtre OpenGL permet une visualisation des obstacles et du meilleur chemin, ainsi que d''une entrée en (0,0) et d'une sortie (400,400) sous forme de carré de 20x20 coloré.

Algorithme

Il s'occupe de définir tout d'abord les chemins initiaux (aléatoire) des individus puis détermine leur avancés en fonction des collisions pour calculer les scores de chaque individu. Les meilleurs individus d'une génération sont sélectionnés pour la reproduction mettant en jeu les mutations et les migrations... La classe *Individu* gère les fonctions unitaires et la classe *village* y fait appel pour calculer les meilleurs et les faire se reproduire.

L'algorithme s'arrête une fois qu'une stabilité est atteinte dans l'ensemble des populations (quand aucun meilleur chemin est trouvé sur 50 générations), ou quand le nombre de générations maximaux est atteint.

Pour déterminer les meilleures suites de déplacements on utilise une fonction d'évaluation sur chaque individu afin de leur attribuer un score (la distance moins le nombre de pas divisé par 2). Les 25% meilleurs scores sont sélectionnés pour se reproduire et participeront à la génération suivante.

Distribution

Le rapport client/serveur se fait en RPC. On utilise une approche serveur/clients pour ce projet.

Dans un premier temps, les clients se connectent au serveur, qui leur envoie les données relatives au contexte d'exécution, entre autre, la liste des obstacles sur la map et leur numéro de village. Le serveur gère les appels à registerrpc dans des threads. Quand les N clients voulus sont connectés, on procède à la mise en place du serveur d'échanges :

celui-ci consiste en la mise à disposition d'un tableau d'adns, que les clients s'échangeront au fur et à mesure de l'exécution. Le tableau est cyclique et se remplit en permanence sans se vider. Les clients peuvent reprendre les adns qu'ils ont eux-même envoyé sans que cela fragilise la stabilité et la fiabilité du système, à cause des probabilités qu'ils le fassent (les clients tendent rapidement vers des chemins dont le nombre de pas est relativement identique).

Algorithmes principaux

Nous allons brièvement présenter l'algorithme de reproduction, qui est le plus important dans une programmation génétique avec la fonction d'évaluation.

Entrée : un village et une fréquence de croisement spécial

Sortie : un village plein de nouveaux habitants

Principe :

On commence par déterminer le nombre d'individus qui vont muter en fonction de la taille de la population et du pourcentage de mutation. On boucle sur les 3 % de meilleurs et sur les 25 % de meilleurs ensuite pour faire se reproduire les 3 % avec les 25 % et obtenir 75 % de nouveaux individus. On lance une pièce pour savoir s'il y a mutation, tant que le nombre de mutations n'est pas atteint (il peut il y en avoir moins que le nombre maximal prévu). Ensuite on a deux types de reproduction, la normale et la fréquence de croisement spécial, qui a 1/4 de chance d'arriver et qui a pour but de reproduire un individu avec un ADN de type affine ou sinusoïde (si l'option convergence assistée est cochée).

La reproduction à proprement parler est un crossing-over de deux ADN qui assemble 10 à 90 % d'un ADN avec le reste du deuxième ADN, l'ordre du premier ADN est choisi aléatoirement pour qu'il y a une meilleure variation dans le brassage. Enfin l'ADN résultant est susceptible de muter, et dans ce cas, une boucle sur le nombre de mutations maximum par ADN sélectionne un élément et le change, aléatoirement dans tout l'ADN. Il n'y a pas de vérification des éléments déjà mutés donc le nombre effectif d'élément muté peut être inférieur au nombre maximal si un élément est tiré deux fois.

Finalement seuls les 25 % de meilleurs restent dans la nouvelle population et en forment une nouvelle avec les 75 % de nouveau-nés.

Répartition du travail

Arthur: *Interface / map / dessin / chemins spéciaux / village / rapport / Réflexions et choix des algorithmes*

Lansiné *: Évaluation / score / détection de collision d'obstacle / UML / Réflexions et choix des algorithmes*

Nicolas: *RPC / distribution / choix des structures / village / Réflexions et choix des algorithmes*

Archade *: mutation / reproduction / tests / village / rapport / Réflexions et choix des algorithmes*

Conclusion

Un projet très intéressant de par la programmation génétique, qui par ailleurs peut être utilisée dans tous les domaines donc le projet se révèle très formateur. Des algorithmes rapides à mettre en place dans leur formes basiques et pourtant qui demandent beaucoup de temps d'affinage pour s'adapter parfaitement au calcule demandé.

La partie distribuée nous offre une démonstration de la puissance des systèmes distribués à grande échelle et nous ne pouvons qu'imaginer leur efficacité sur d'autres algorithmes génétiques plus conséquents, déjà capables de rivaliser et de surpasser, en terme de résultats et d'optimisation, les solutions standards existantes.