# Rapport PDI

## Choix de modélisation

### Présentation

Les clients se connectent au serveur, celui-ci va sauvegarder dans une liste le client afin de pouvoir communiquer correctement avec lui. Une fois connecté au serveur, le client se met en attente d’un appel RPC.

Le serveur découpe la grille en fonction du nombre de clients, et retourne une liste de grilles.

Chaque grille dans cette liste sera envoyée aux différents clients.

Le serveur exécute un thread pour chaque client, il attend le retour de callrpc.

Une fois qu’un client réceptionne une grille, il la fait évoluer et la renvoie au serveur.

Le serveur intègre la grille renvoyée par le client dans la grille principale présente sur le serveur (celle qui est affichée).

Une fois que tous les clients ont renvoyé leur grille, le serveur la redécoupe et la redistribue aux clients disponibles et ainsi de suite.

Ce système permet à plusieurs clients de se connecter et se déconnecter quand ils veulent, le système s’adaptera au nombre de clients.

### Découpage

En premier lieu, l’algorithme de découpage découpait des paquets de cellules. Il parcourait la grille en suivant les cellules proches l’une de l’autre, ce qui constituait une matrice ignorant les cellules mortes. On obtenait ainsi la plus petite matrice possible contenant des cellules vivantes. Cette méthode prenait en compte la densité de population.

L’inconvénient de cette méthode est que premièrement, si la grille est entièrement remplie de cellules vivantes, ou bien qu’elles sont toutes suffisamment proches, l’algorithme retournait la grille complète. Deuxièmement, l’algorithme ne prenait pas en compte le nombre de clients mais le nombre de paquets de cellules isolés.

Pour découper la grille, nous faisons un découpage sous forme de « trames », en fonction du nombre de clients, elle est découpée verticalement. Une trame est une matrice de largeur variable, mais de même hauteur que la grille principale. Chaque trame est mise dans une liste qui sera traitée par le serveur.

### Communication

Pour communiquer entre le serveur et les clients, nous avons opté pour le protocole UDP. Ce choix s’est basé sur sa légèreté par rapport au protocole TCP. Il permet un envoi de paquets plus petit ce qui optimise les temps de réponse des différents clients, et donc la vitesse de calcul globale.

Nous avions hésité avec TCP car il était plus simple de détecté un client qui se déconnecte et de réagir en conséquence.

## Algorithmes principaux (pseudo-code)

### Algorithme de découpe de la grille

explode\_grille : Grille g, entier nbDecoupe

DEBUT

tailleTrame = ceil(largeur(g) / nbDecoupe)

debut = 0 ; fin = tailleTrame;

Liste l = nouvelleListe()

pour i = 0 jusqu'à i < nb pas de 1

grilleTemporaire = couper(g, 0, hauteur(g), debut, fin)

capsule = {grilleTemporaire, 0, debut, NON\_TRAITER} // Structure coupe\_capsule

debut += tailleTrame

fin += tailleTrame

ajoutListe(l, capsule)

fpour

FIN

### Algorithme de découpe dans la grille à partir de coordonnées debut/fin

Couper : Grille g, entier i\_deb i\_fin j\_deb j\_fin

DEBUT

Grille nouvelleGrille = init(i\_fin – i\_deb + 2, j\_fin – j\_debut + 2)

i\_deb- - ; i\_fin ++ ; j\_deb - - ; j\_fin ++

pour i=0, ig=i\_deb jusqu'à i < nouvelleGrille->hauteur pas de 1

pour j=0, jg=j\_deb jusqu'à j < nouvelleGrille->largeur pas de 1

si interieur\_du\_plateau(i, j) alors

etat = 0

else

etat = g->plateau[ig][jg].etat

fsi

nouvelleGrille[i][j].etat = etat

fpour

fpour

retourne nouvelleGrille

FIN

### Algorithme de reconstruction de la grille principale depuis une liste

implode\_grille : Grille g, Liste l

DEBUT

coupe\_capsule tmp;

tant que (tmp = liste\_suiv(l)) != NULL faire

coller(g, tmp->g, tmp->x, tmp->y)

fintantque

return g

FIN

### Algorithme de collage d'une grille dans la grille principale

coller : Grille origine, Grille g, entier : x, y

DEBUT

pour i=x, ig=1 jusqu'à i < x + g->i\_fin - g->i\_debut - 1 pas de 1

pour j=y, jg=1 jusqu'à j < y + g->j\_fin - g->j\_debut - 1 pas de 1

si interieur\_du\_plateau(i, j) alors

origine->plateau[i][j] = g->plateau[ig][jg].etat

finsi

finpour

finpour

FIN

### Algorithme d'évolution de la grille

evolution : Grille g

DEBUT

pour i=1 jusqu'à i<hauteur(g) - 1 pas de 1

pour j=1 jusqu'à j<largeur(g) - 1 pas de 1

g->plateau[i][j].petat = g->plateau[i][j].etat

pour k=i-1 jusqu'à k<=i+1 pas de 1

pour l = j-1 jusqu'à l<=j+1 pas de 1

si !(i==k && j==l) alors

si g->plateau[k][l].etat == 1 alors

nombreDeCellules++

finsi

finsi

finpour

finpour

si g->plateau[i][j].etat == 1 alors

si nombreDeCellules > 3 ou nombreDeCellules < 2 alors

g->plateau[i][j].petat = 0 // La cellule meurt

finsi

sinon si g->plateau[i][j].etat == 0 alors

si nombreDeCellules == 3 alors

g->plateau[i][j].petat = 1 // La cellule nait

finsi

finsi

finpour

finpour

pour i=1 jusqu'à i<hauteur(g) - 1 pas de 1

pour j=1 jusqu'à j<largeur(g) - 1 pas de 1

g->plateau[i][j].etat = g->plateau[i][j].petat

finpour

finpour

return g

FIN

## Répartition du travail

* Gosset Thomas
  + Création de l’algorithme du jeu de la vie
    - Création de la grille par lecture de fichier texte
    - Algorithme d'évolution des cellules
  + Communication entre clients et serveur
    - Liaison client / serveur via UDP
    - Communication RPC
  + Créations de threads sur le serveur
    - Création d'un thread pour chaque appel RPC
* Clog Maxime
  + Création des listes et files de grilles
    - Structures de données et stockage des grilles
  + Algorithmes de découpage / ré-assemblage de la grille
    - Création du premier algorithme de découpe
    - Création du second algorithme de découpage
      * Débogage et correction par Thomas