ALGORITHMES DES RESEAUX

Projet C-TRON

06 DECEMBRE 2022

HAYK ZARIKIAN ET ALEXANDRE DUBERT

Table des matières

1 Introduction 1
2 Client 1
2.1 Connexion TCP avec le serveur 1
2.2 Envois des informations au serveur 1
2.3 Réception des informations envoyées par le serveur 3
2.4 Affichage du plateau 3
3 Serveur 4
3.1 Connexion TCP avec le(s) client(s) 4
3.2 Envois des informations aux clients 4
3.3 Réception des informations envoyées par les clients 4
3.4 Construction du plateau 6
4 Tests / Compilation / Exécution 7
5 Questions 7
5.1 Quels sont les avantages et inconvénients de ce genre d'architecture ? Examinez l'impact des caractéristiques de la connexion. 7
5.2 L'utilisation du protocole TCP est-elle souhaitable pour les jeux en ligne ? Qu'en est-il du *Cloud Gaming* ? Quel(s) protocoles utiliseriez-vous dans le cas présent ? 7
6 Conclusion 8

1 Introduction

Ce rapport est dédié à notre projet d'algorithmes des réseaux dans le cadre de notre troisième année de licence d'Informatique à l'UFR de Mathématique et d'Informatique à l'Université de Strasbourg. L'objectif de ce projet était d'élaborer une connexion entre le(s) client(s) et le serveur avec le protocole TCP afin de permettre l'accès au jeu « Tron » programmé en langage C avec la librairie « ncurses ». Pour ce faire, nous avons choisi de travailler en binôme par l'intermédiaire d'un répertoire GIT sur le GitLab de l'Université. Nous nous sommes réparti les tâches au préalable, et dès que nous finissions une étape importante, nous l'envoyons sur le dépôt distant.

2 Client

2.1 Connexion TCP avec le serveur

La première étape pour la programmation du fichier « client.c » est d'établir la connexion fiable TCP avec le serveur. Le fichier « client.c » doit permettre deux types de connexions, la première est la connexion en locale, la seconde est la connexion en réseau. Pour ce faire, nous avons d'abord décidé de programmer le client sans utiliser la librairie « ncruses » afin de mieux gérer les potentielles erreurs et de mieux les détecter. Nous avons également utilisé la fonction `select()` pour mettre en commun plusieurs descripteurs de fichiers notamment l'entrée standard pour l'envoie des informations et le descripteur du fichier du socket du serveur pour la réception des informations.

2.2 Envois des informations au serveur

Le client envois les informations du clavier (changement de direction) au serveur dès la détection d'une pression sur une touche du clavier valide. Pour vérifier si la touche est conforme, nous utilisons la fonction `change_direction()`. (Figure 1)

```
75          /*
76              Envois des informations (structure client_input) de direction vers le serveur.
77          */
78          if (FD_ISSET (0, &tmp))
79          {
80              ssize_t n_lus;
81              n_lus = read (0, &touche_clavier, 1);
82              CHECK (n_lus != -1);
83              if (change_direction (touche_clavier, &input, nb_joueur, ip_serveur)) {
84                  CHECK (send (sockfdServeur, &input, sizeof (struct client_input), 0) != -1);
85              }
86          }
```

Figure 1: Envois des informations au serveur

La fonction `change_direction()` prend en argument la touche du clavier, la structure input, le nombre de joueur et l'adresse IP. Nous allons utiliser un switch afin de déterminer si la touche appuyée fait partie de la liste {'z', 'q', 's', 'd', ' '}, si c'est le cas la valeur de retour sera de 1 sinon de 0 afin d'exclure l'envoie des données pour une touche non valide. Nous allons également prendre en compte si le nombre de joueur vaut 2 et si le jeu se passe en local alors les touches {'i', 'j', 'k', 'l', 'm'} seront prises en comptes contrairement au jeu en réseau (claviers différents). (Figure 2 et 3)

```c
/**
 * @brief Fonction qui récupère la touche appuyé pour insérer la direction dans la structure
 *        input avec l'identificant du joueur.
 *
 * @param[in] touche_clavier char : Touche appuyée
 * @param[in] input struct client_input : Direction enregistrée avec l'identifiant du joueur
 * @param[in] nb_joueur int : Nombre de joueur
 * @param[in] ip_serveur char* : IP du serveur
 * @returns[in] input struct client_input : Affectation des valeurs par adresse de la structure input
 * @return[out] int : Retourne 1 si une touche valide a été appuyée
 */
int change_direction (char touche_clavier, struct client_input *input, int nb_joueur, char* ip_serveur)
{
    switch (touche_clavier)
    {
    case 'z':
        input->id = 1;
        input->input = UP;
        return 1;
    case 'q':
        input->id = 1;
        input->input = LEFT;
        return 1;
    case 's':
        input->id = 1;
        input->input = DOWN;
        return 1;
    case 'd':
        input->id = 1;
        input->input = RIGHT;
        return 1;
    case ' ':
        input->id = 1;
        input->input = TRAIL_UP;
        return 1;
    }
```

Figure 2: Fonction change_direction () (1)

```c
    if (nb_joueur == 2 && !strcmp (ip_serveur, "0.0.0.0"))
    {
        switch (touche_clavier)
        {
        case 'i':
            input->id = 2;
            input->input = UP;
            return 1;
        case 'j':
            input->id = 2;
            input->input = LEFT;
            return 1;
        case 'k':
            input->id = 2;
            input->input = DOWN;
            return 1;
        case 'l':
            input->id = 2;
            input->input = RIGHT;
            return 1;
        case 'm':
            input->id = 2;
            input->input = TRAIL_UP;
            return 1;
        }
    }
    return 0;
}
```

Figure 3 : Fonction change_direction () (2)

2.3 Réception des informations envoyées par le serveur

La réception des informations envoyées par le serveur se passe lorsque le serveur envois un `send()` au(x) client(s), ainsi celui-ci peut réception avec `recv()`. La récupération du plateau permet donc l'affichage du jeu via l'interface graphique « ncurses ». Nous avons utilisé la fonction d'affichage `display_character()` et les macros fournit dans le template. (Figure 4)

```
87          /*
88              Réception des informations du plateau de jeu (structure display_info) du serveur.
89          */
90          if (FD_ISSET (sockfdServeur, &tmp))
91          {
92              memset(&plateau_jeu, '\0', sizeof(display_info));
93              CHECK (recv (sockfdServeur, &plateau_jeu, sizeof (display_info), 0) != -1);
94              clear ();
95              for (int x = 0; x < XMAX; x++)
96                  for (int y = 0; y < YMAX; y++)
97                  {
98                      if (x == 0 || x == XMAX-1)
99                      {
100                         if (plateau_jeu.board[x][y] == WALL)
101                             display_character (WALL, y, x, ACS_VLINE);
102                     }
103                     else
104                     {
105                         if (plateau_jeu.board[x][y] == BLUE_ON_BLUE)
106                             display_character (BLUE_ON_BLUE, y, x, ACS_HLINE);
107                         else if (plateau_jeu.board[x][y] == YELLOW_ON_YELLOW)
108                             display_character (YELLOW_ON_YELLOW, y, x, ACS_HLINE);
109                         else if (plateau_jeu.board[x][y] == WALL)
110                             display_character (WALL, y, x, ACS_HLINE);
111                         else if (plateau_jeu.board[x][y] == BLUE_ON_BLACK)
112                             display_character (BLUE_ON_BLACK, y, x, 'O');
113                         else if (plateau_jeu.board[x][y] == YELLOW_ON_BLACK)
114                             display_character (BLUE_ON_BLACK, y, x, 'O');
115                     }
116                 }
117             mvaddstr (0, XMAX/2 - strlen("C-TRON")/2, "C-TRON");
118             refresh();
119         }
120     }
```

Figure 4 : Réception des informations envoyées par le serveur

2.4 Affichage du plateau

Comme dit dans la partie précédente, l'affichage du plateau se passe à la réception de la structure `display_info` par le serveur. Cependant, lors de la terminaison du jeu suite à la victoire d'un des deux joueurs ou aucun des joueurs, la fenêtre quitte proprement « ncruses » en affichant le résultat de la partie. (Figure 5)

```
121     /*
122         Affichage du vainqueur et fermeture du jeu.
123     */
124     clear ();
125     if (plateau_jeu.winner == 0)
126         mvaddstr (YMAX/2, XMAX/2 - strlen("END GAME : No winner")/2, "END GAME : No winner");
127     else {
128             mvaddstr (YMAX/2, XMAX/2 - strlen("END GAME : Winner is ")/2, "END GAME : Winner is ");
129             printw ("%d", plateau_jeu.winner);
130     }
131     refresh ();
132     sleep (3);
133
134     endwin ();
135     CHECK (close (sockfdServeur) != -1);
136     return EXIT_SUCCESS;
```

Figure 5 : Affichage et fermeture du jeu

3 Serveur

3.1 Connexion TCP avec le(s) client(s)

Pour commencer, nous paramétrons l'adresse du serveur et nous attachons à un socket `bind()`. Ensuite nous écoutons le socket `listen()`, nous permettons au maximum deux connexions. Puis, nous attendons la connexion du premier client `accept()`. Lorsqu'il est connecté, il envoie les informations initiales. S'il est le seul sur son terminal, nous le considérons comme le joueur 1 et nous faisons de même pour le deuxième client, sinon nous ignorons les demandes de connexion. Dans la suite du programme, nous testons la présence du deuxième client avec la valeur de son socket, -1 s'il n'est pas connecté. Enfin, nous initialisons le plateau et les « fdsets ».

3.2 Envois des informations aux clients

À chaque tour de boucle, nous faisons un `send()` du `display_info` à destination des clients. La mise à jour du plateau est décrite dans la partie 3.4. (Figure 6)

```
292 void fin_propre(int s_srv, int s_cl1, int s_cl2, display_info *di, int gagnant) {
293     di->winner = gagnant;
294     CHECK(send(s_cl1, di, sizeof(struct display_info), 0) != -1);
295     if (s_cl2 != -1)
296         CHECK(send(s_cl2, di, sizeof(struct display_info), 0) != -1);
297     close(s_cl1);
298     if (s_cl2 != -1) close(s_cl2);
299     close(s_srv);
300 
301     exit(EXIT_SUCCESS);
302 }
```

Figure 6 : Fonction qui permet l'envoie des informations au(x) client(s)

3.3 Réception des informations envoyées par les clients

Comme nous écoutons potentiellement deux sockets et l'entrée standard du serveur, nous devons utiliser la fonction `select()`. Si dans l'intervalle de temps du « timer refresh » des données sont reçues sur l'un des « fdset », alors nous effectuons le traitement adéquat, sinon nous mettons à jour le plateau et nous recommençons la boucle. Si les données reçues viennent de l'une des sockets, si c'est une direction, nous vérifions qu'elle est valide (pas de demi-tour), puis nous stockons l'entrée dans une structure « player_info » qui contient toutes les informations d'un joueur. Si les données viennent de l'entrée standard, nous interprétons la commande et nous effectuons le traitement demandé. Nous avons rencontré une difficulté avec select, en effet rien ne fonctionnait alors que tout semblait correct au niveau algorithmique. Finalement, après trois heures nous nous sommes rendu compte que le select modifiait le timer et qu'il fallait donc réinitialiser le timer à chaque tour de boucle. (Figure 7, 8 et 9)

```
82          /* Données dans l'entrée standard */
83          if (FD_ISSET(0, &tmp)) {
84              memset(entree_term, '\0', 10);
85              CHECK(read(0, entree_term, 10) != -1);
86              if (!strcmp(entree_term, "quit\n")) {
87                  fin_propre(s, s_client1, s_client2, &di, 0);
88              } else if (!strcmp(entree_term, "restart\n")) {
89                  init_game(&di, &p1_info, &p2_info);
90              } else if (!strcmp(entree_term, "help\n")) {
91                  printf("restart : recommence la partie.\n");
92                  printf("quit : quitte le jeu.\n");
93              } else {
94                  printf("Commande inconnue, \"help\" pour obtenir de l'aide.\n");
95              }
96          }
```

Figure 7 : FDSET entrée standard

```
98          /* Données dans la socket du client 1 */
99          if (FD_ISSET(s_client1, &tmp)) {
100             memset(&cl_in, '\0', sizeof(struct client_input));
101             CHECK(recv(s_client1, &cl_in, sizeof(struct client_input), 0) != -1);
102
103             if (s_client2 == -1 && cl_in.id == 2) {
104                 /* Traitement de l'entrée du joueur 2 (si 2 joueurs sur 1 terminal) */
105                 if (cl_in.input == TRAIL_UP) {
106                     p2_info.trail = (p2_info.trail + 1) % 2;
107                     if (p2_info.trail) remettre_trainee(&di, 2);
108                 } else {
109                     dir = mise_a_jour_direction(p2_info.dir, (int) cl_in.input);
110                     p2_info.dir = dir;
111                 }
112             } else {
113                 /* Traitement de l'entrée du joueur 1 */
114                 if (cl_in.input == TRAIL_UP) {
115                     p1_info.trail = (p1_info.trail + 1) % 2;
116                     if (p1_info.trail) remettre_trainee(&di, 1);
117                 } else {
118                     dir = mise_a_jour_direction(p1_info.dir, (int) cl_in.input);
119                     p1_info.dir = dir;
120                 }
121             }
122         }
```

Figure 8 : FDSET client 1

```
124         /* Données dans la socket du client 2 */
125         if (s_client2 != -1 && FD_ISSET(s_client2, &tmp)) {
126             memset(&cl_in, '\0', sizeof(struct client_input));
127             CHECK(recv(s_client2, &cl_in, sizeof(struct client_input), 0) != -1);
128
129             /* Traitement de l'entrée du joueur 2 */
130             if (cl_in.input == TRAIL_UP) {
131                 p2_info.trail = (p2_info.trail + 1) % 2;
132                 if (p2_info.trail) remettre_trainee(&di, 2);
133             } else {
134                 dir = mise_a_jour_direction(p2_info.dir, (int) cl_in.input);
135                 p2_info.dir = dir;
136             }
137         }
138
139         /* Mise à jour du plateau et de la condition d'arrêt */
140         gagnant = mise_a_jour_board(&di, &p1_info, &p2_info);
141     }
142
143     fin_propre(s, s_client1, s_client2, &di, gagnant);
144
145     return EXIT_SUCCESS;
```

Figure 9 : FDSET client 2

3.4 Construction du plateau

Le plateau est initialisé avec des murs (constante `WALL`), les têtes des joueurs (constante `HEAD` / `HEAD` + 1) et des cases vides (constante `EMPTY`). Ensuite, nous déplaçons la tête des joueurs en fonction de leur direction respective. L'ancienne position de la tête devient une traînée (constante `TRAIL_INDEX_SHIFT` / `TRAIL_INDEX_SHIFT` + 1) si la traînée est activée, sinon la constante `TRAIL_DOWN` / `TRAIL_DOWN` - 1 (pour pouvoir la réactiver ensuite). Avant de mettre à jour la position, nous vérifions qu'il n'y a pas de collision, s'il y en a une, nous définissons le gagnant dans le `display_info` pour que la partie puisse terminer côté client. (Figure 10, 11 et 12)

```
148 void init_game (display_info *di, player_info *pi1, player_info *pi2) {
149     int i, j;
150 
151     pi1->id = 1;
152     pi1->dir = RIGHT;
153     pi1->posx = XMAX / 3;
154     pi1->posy = 2 * YMAX / 3;
155     pi1->trail = 1;
156 
157     pi2->id = 2;
158     pi2->dir = LEFT;
159     pi2->posx = 2 * XMAX / 3;
160     pi2->posy = YMAX / 3;
161     pi2->trail = 1;
162 
163     for (j = 0; j < XMAX; j++) {
164         for (i = 0; i < YMAX; i++) {
165             if (i == 0 || i == YMAX-1 || j == 0 || j == XMAX-1) {
166                 di->board[j][i] = WALL;
167             } else {
168                 di->board[j][i] = -1;
169             }
170         }
171     }
172 
173     di->board[pi1->posx][pi1->posy] = 0;
174     di->board[pi2->posx][pi2->posy] = 1;
175 
176     di->winner = -1;
177 }
```

Figure 10 : Fonction initialisation jeu

```
198 int mise_a_jour_board(display_info *di, player_info *pi1, player_info *pi2) {
199     int p1_oldx, p1_oldy, p2_oldx, p2_oldy;
200     int gagnant;
201 
202     p1_oldx = pi1->posx;
203     p1_oldy = pi1->posy;
204 
205     switch (pi1->dir) {
206         case UP:
207             pi1->posy = p1_oldy - 1;
208             break;
209         case DOWN:
210             pi1->posy = p1_oldy + 1;
211             break;
212         case RIGHT:
213             pi1->posx = p1_oldx + 1;
214             break;
215         case LEFT:
216             pi1->posx = p1_oldx - 1;
217             break;
218     }
219 
220     p2_oldx = pi2->posx;
221     p2_oldy = pi2->posy;
```

Figure 11 : Fonction de mise à jour du plateau (1)

```
223     switch (pi2->dir) {
224         case UP:
225             pi2->posy = p2_oldy - 1;
226             break;
227         case DOWN:
228             pi2->posy = p2_oldy + 1;
229             break;
230         case RIGHT:
231             pi2->posx = p2_oldx + 1;
232             break;
233         case LEFT:
234             pi2->posx = p2_oldx - 1;
235             break;
236     }
237
238     gagnant = test_collision(di, pi1, pi2);
239     if (gagnant == -1) {
240         di->board[pi1->posx][pi1->posy] = 0;
241         di->board[pi2->posx][pi2->posy] = 1;
242         di->board[p1_oldx][p1_oldy] = pi1->trail == 1 ? TRAIL_INDEX_SHIFT : TRAIL_DOWN;
243         di->board[p2_oldx][p2_oldy] = pi2->trail == 1 ? TRAIL_INDEX_SHIFT + 1 : TRAIL_DOWN - 1;
244     }
245
246     return gagnant;
247 }
```

Figure 12 : Fonction de mise à jour (2)

4 Tests / Compilation / Exécution

Nous avons effectué tous les tests nécessaires pour le bon fonctionnement du jeu à chaque étape de notre programmation (test de collision, test du jeu en réseau, test du jeu en local, test des fonctionnalités, …). Nous fournirons un fichier « makefile » qui nous a permis de faciliter la compilation. Nous avons également décidé de fournir les exécutables (client et serveur) dans le cas d'une impossibilité de compilation. Attention, la compilation nécessite la librairie « ncruses » et peut se faire avec la commande « make ».

5 Questions

5.1 Quels sont les avantages et inconvénients de ce genre d'architecture ? Examinez l'impact des caractéristiques de la connexion.

Dans notre projet, nous utilisons le protocole TCP orienté connexion dont ses avantages sont les suivants. La connexion est fiable, la détection d'erreurs et de corrections, le contrôle de congestion, transmission d'un accusé de réception, le protocole est sécurisé, taux de perte de paquets plus faible. Cependant, ce protocole possède aussi des inconvénients comme une vitesse plus basse, plus compliqué à mettre en place, vulnérable à une attaque par déni de service.

5.2 L'utilisation du protocole TCP est-elle souhaitable pour les jeux en ligne ? Qu'en est-il du *Cloud Gaming* ? Quel(s) protocoles utiliseriez-vous dans le cas présent ?

D'après la question précédente, nous pouvons nous demander si cela est vraiment nécessaire d'avoir une connexion fiable (donc par TCP) pour les jeux en ligne. Prenons l'exemple d'un jeu dans lequel le client reçoit la position des autres joueurs depuis un serveur. Si un paquet se perd, attendre sa retransmission ferait perdre du temps et augmenterai donc la latence, alors qu'ignorer la perte ne se remarquerait presque pas comme les paquets suivants modifieraient à nouveau la position des joueurs. C'est

pourquoi nous pensons qu'un protocole sans retransmission comme UDP ou RTP est préférable à TCP. Il en va de même pour le *Cloud Gaming*. Cette fois le serveur ne se contente pas d'envoyer les positions des joueurs, il calcule toutes les données, les traites et envoi aux clients le rendu graphique. Le flux de données est considérable (15 Mbits/s pour PS Now et 45 Mbits/s pour Stadia et GeForce Now), il est donc encore bien plus important de privilégier la vitesse à la fiabilité. En outre, si une image est transmise avec une erreur ce n'est pas toujours si grave car le paquet suivant sera sûrement transmis sans ce défaut, le client ne sera donc pas dérangé par cette erreur.[1]

6 Conclusion

Ce projet nous a beaucoup apporté sur trois plans. Premièrement sur le plan de la pratique dans l'algorithmes des réseaux. En effet, il a fallu s'approprier la programmation réseaux, la comprendre, trouver les moyens d'obtenir une connexion fiable entre le client et le serveur. Deuxièmement sur le plan de la pratique d'une nouvelle librairie « ncruses » dont nous avons pu découvrir les possibilités qu'elles nous offraient et rendre le projet plus vivant. Et dernièrement sur le plan du travail d'équipe. Le partage du travail était très équilibré, la communication très bonne. Nous nous sommes mutuellement donnés des idées et conseillés. L'utilisation d'un dépôt GIT était une bonne idée et nous a fait gagner en efficacité. Finalement, ce projet a été très plaisant à mener à bien et riche, le résultat est très satisfaisant et nous avons beaucoup progressé dans le domaine de la programmation d'algorithmes des réseaux.

[1] A Network Analysis on *Cloud Gaming* Stadia, GeForce Now and PS Now