Rapport de projet : Jeu de Dames en réseau

Anthony Bertrand, Quentin Bandera

Contents

1 Introduction

Lors de nos études en L3 Informatique, nous avons dû implémenter un jeu de Dame en réseau. Le but de ce projet est de mettre en pratique les connaissances acquises aussi bien en cours magistral qu'en travaux pratiques. Nous allons tout d'abord revenir sur les règles à implémenter pour le jeu de Dame. Nous verrons ensuite quels sont les choix à notre disposition en ce qui concerne les protocoles réseaux, ainsi que les structure que nous allons utiliser. Enfin, nous regarderons en détail le travail effectué ainsi que les potentielles améliorations.

2 Le jeu de Dame

2.1 Origine

Le jeu de dame (Draughts ou Checkers en anglais) est un jeu pour deux joueurs se jouant avec un plateau. Les origines du jeu viendraient de l'Egypte antique, vers -1500 avant Jésus Christ.

2.2 Règles du jeux

2.2.1 But du jeu

Le but du jeu de dames est de prendre toutes les pièces de l'adversaire. Pour se faire, les joueurs disposent de 20 pièces disposées sur le plateau et séparées d'une case. On joue généralement sur les cases noires (ref Figure 1). La taille du plateau peut varier suivant les règles mais la version du jeu la plus connue est celle qui se joue sur un plateau 10x10 cases.

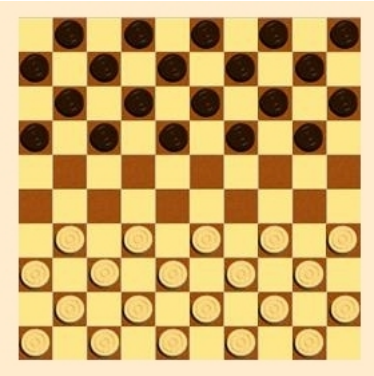

Figure 1: plateau de jeu

2.2.2 Le déplacement des pièces

Lors de son tour d'action, le joueur peut déplacer l'un de ses pions vers une case noire adjacente (il effectue donc un déplacement diagonal). Le joueur ne peut pas faire reculer ses pièces, il ne peut se déplacer qu'en avant (sauf pour les prises). Si une pièce est au bord droit ou gauche du plateau, elle n'a donc qu'un seul déplacement disponible. Le joueur ne peut pas déplacer sa pièce sur une case déjà occupée par une pièce. (ref Figure 2)

2.2.3 Le déplacement des dames

Le déplacement d'une dame est beaucoup moins limité. Ces dernières peuvent se déplacer en avant et en arrière. Elle peuvent également se déplacer d'autant de cases qu'elle veulent, du moment qu'elle ne rencontre pas d'obstacle (une autre pièce ou le bord du plateau). (ref Figure 2)

Figure 2: déplacements

2.2.4 La prise avec des pièces

Pour prendre une pièce de l'adversaire, il faut que l'une des pièces du joueur soit adjacente à une pièce de l'adversaire et que la case derrière cette dernière soit vide. Le but lors des déplacements de vos pièces sera donc d'éviter à tout prix de laisser une pièce sans défenses. La prise en arrière est possible ! (ref Figure 3)

2.2.5 La prise avec des dames

Comme pour les déplacements, la dame n'a pas besoin d'être adjacente au pion qu'elle veut prendre. Elle peut se déplacer d'autant de cases qu'elle veut du moment qu'elle ne rencontre pas d'obstacle entre elle et le pion, et du moment que la case derrière le pion est vide. Si les autres cases derrière le pion sont aussi vides, elle peut choisir de s'arreter sur l'une d'elles. (ref Figure 4)

2.2.6 Les rafles

Après avoir prit une pièce de l'adversaire, le joueur peut continuer de prendre d'autre pièces avec le même pion s'il en a la possibilité (et il le doit !). La rafle est beaucoup plus simple avec une dame de par ses mouvements moins restreints (ref Figure 3 et 4)

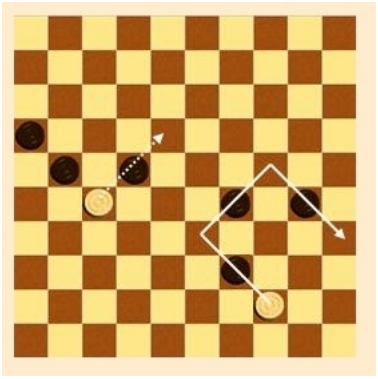

Figure 3: prise simple et rafle

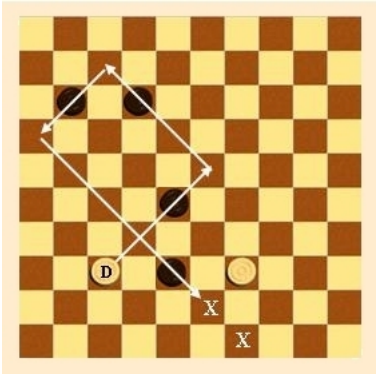

Figure 4: rafle avec une dame

2.2.7 Création des dames

Lorsqu'une pièce arrive de l'autre coté du plateau, sur la toute dernière rangé, et qu'elle ne peut plus avancer, elle devient une dame. Comme vu précédemment, les dames ont une plus grande liberté de déplacement et peuvent se déplacer en arrière. Il est donc dans l'interêt du joueur de créer des dames en empêchant la création de dames adverses.

3 Etude du problème

3.1 Partie réseau

3.1.1 Choix du protocole de transport

Dans le monde du jeu vidéo en ligne, le protocole UDP est très souvent utilisé. UDP possède une faible latence pour le traitement ce qui est l'élement recherché dans des jeux nerveux comme Overwatch, League of Legends, Unreal Tournament etc... Dans notre cas, le jeu se fera en tour par tour. Notre choix se porte donc sur TCP car nous ne voulons pas de perte de paquet lorsque le joueur valide son coup.
Nous aurions pu utiliser un protocole applicatif existant, se basant sur UDP, voire même créer notre propre protocole mais par soucis de temps, nous nous contenterons de TCP.

3.2 Partie algorithmique du jeu de dames

3.2.1 Structures utilisées

Afin de mener à bien ce projet, nous avons décidé de ne pas utiliser de structures trop complexes. Le plateau de jeu est représenté par un tableau à une dimension. Nous simulons la deuxième dimension du tableau en utilisant une variable i, j et une constante pour le modulo. Une case du plateau avec un tableau à deux dimensions ressemblerai à $plateau[i][j]$. Avec notre implémentation, elle ressemble à $plateau[i * CONSTANTE + j]$, la constante étant le nombre de conlonne du plateau.
Nous avons défini une structure *Pion* avec une coordonée x et une coordonnée y afin de pouvoir transmettre facilement des positions sur le plateau au serveur. Enfin, nous avons utilisé des constantes définies dans le fichier dame.h afin de faciliter la lecture de code ainsi que les éventuels changements.

```
#define PION_BLANC 2
#define DAME_BLANC 4
#define PION_NOIR 1
#define DAME_NOIR 3
#define CASE_VIDE 0
#define TAILLE_PLATEAU 10
#define JOUEUR_BLANC 0
#define JOUEUR_NOIR 1

typedef struct pion
{
    int x;
    int y;
}Pion;
```

4 Travail effectué

Avant de commencé cette partie, nous tenons à préciser que ce projet a dû être réalisé en deux semaine, par des étudiants ayant deux mois d'expérience dans le domaine du réseau. Le temps étant la ressource qui nous fit le plus défaut, nous avons décidé de nous concentrer au maximum sur la partie réseau et non sur la partie algorithmique du jeu de Dame.

4.1 Echanges TCP

Comme dit plus haut, nous avons décidé d'utiliser le protocole TCP au niveau de la couche de transport. Nous allons présenter dans cette section les algorithmes implémentés pour la gestion du protocole TCP.

4.1.1 Le serveur

Dans notre serveur, nous utilisons un tableau d'entier $connfd[2]$ afin de stocker les adresses de nos deux clients. Nous créons ensuite la socket et la stockons dans $sockfd$.

```
//création de socket
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    perror("problème lors de la création de la socket");
    exit(-1);
}
```

Nous configurons ensuite notre structure $sockaddr\_in$ pour le TCP :

```
servaddr.sin_family = AF_INET; // IPv4
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(atoi(argv[1]));
```

Nous lions ensuite la socket à l'adresse IP :

```
//lie la socket à l'adresse IP
if(bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
{
    perror("encore rate");
    exit(-1);
}
```

Nous mettons notre serveur sur écoute afin de pouvoir accepter des clients :

```
//serveur écoute
if((listen(sockfd, 1)) != 0)
{
    printf("Listen failed...\n");
    exit(0);
}
printf("Server listening..\n");
```

Puis nous acceptons nos deux clients :

```
//accepte les clients
for (int i = 0; i < 2; i++)
{
    if((connfd[i] = accept(sockfd, (struct sockaddr *)&cliaddr, &len)) < 0)
    {
        printf("server acccept failed...\n");
        exit(0);
    }
    else
        printf("server acccept the client %d\n", i+1);
}
```

Ces étapes préliminaires sont indispensables pour que les clients communiquent avec le serveur. Après avoir établie la connection entre les clients et le serveur, nous initialisons le plateau de jeu et nous commençons la boucle de jeu

```
init_game(plateau);
//game loop
while(1)
{
    for (int i = 0; i < 2; i++)
    {
        write(connfd[i], (const char*)&plateau, sizeof(plateau));
        printf("le plateau....\nJoueur %d reflechi...\n", i+1);
        write(connfd[i], (const char*)&buf, sizeof(buf));
        recv(connfd[i], &p, sizeof(p), 0);
        printf("Coordonnées envoyées par le j%d :\n x : %d\ny : %d\n", i+1, p.x, p.y);
    }
}
```

Dans cette boucle de jeu, le serveur commence par jouer avec le joueur 1. Il lui envoie le plateau de jeu puis lui demande de choisir le pion à déplacer. Ensuite, il reçoit les coordonnées envoyées par le joueur 1 avant de passer au joueur 2.
Nous n'avons pas implémenté les déplacements dans le serveur par manque

de temps, mais nous affichons les structures reçues afin de montrer que les échanges sont opérationnels. De plus, nous ne demandons qu'une seule paire de coordonées alors qu'il nous en faudrai une seconde (pour la destination du coup à jouer).

4.1.2 Le client

Comme pour le serveur, le client doit passer par plusieurs étapes afin d'établir une connection. Il doit dans l'ordre créer la socket, configurer sa structure *sockaddr_in* (son adresse) puis se connecter au serveur TCP.

```
int sockfd, connfd, plateau[TAILLE_PLATEAU*TAILLE_PLATEAU];
char msg_cli[100];
Pion p;
struct sockaddr_in servaddr;

if(argc != 3)
{
    printf("utilisation   : ./client [adresse] [port]\n");
    return 0;
}

// creation du file descriptor de socket
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
    perror("rate");
    exit(-1);
}

// remplissage des info du serveur
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(atoi(argv[2]));
inet_aton(argv[1], &servaddr.sin_addr);

if (connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) != 0) {
    printf("connection with the server failed...\n");
    exit(0);
}
else
    printf("connected to the server..\n");
```

Ensuite, il entame également sa boucle de jeu où il va recevoir le plateau de jeu ainsi que les instructions du serveur. Le joueur doit ensuite saisir les coordonnées du pion à jouer afin que l'algorithme du client les envoie au serveur. Le joueur doit ensuite attendre que l'autre joueur ait fini son tour d'action.

```
while(1){
```

```
        recv(sockfd, &plateau, sizeof(plateau), 0);
        afficher_plateau(plateau);
        recv(sockfd, &msg_cli, sizeof(msg_cli), 0);
        printf("%s\n", msg_cli);
        printf("x : ");
        scanf("%d", &p.x);
        printf("y : ");
        scanf("%d", &p.y);
        /*recevoir le plateau (faire une structure ?)*/
        write(sockfd, (const char*)&p, sizeof(p));
    }
```

4.1.3 Les échanges

Lors de la partie, le serveur envoie régulièrement le plateau de jeu à ses joueurs, ainsi que les consignes à suivre pour ces derniers. Nous envoyons donc un tableau d'entier et des chaines de caractères.
Les clients, eux, envoient une structure *Pion* constituée de la coordonnée x et y de la pièce qu'ils veulent jouer ou de la case destination.

4.2 Le jeu de dames

4.2.1 Initialisation

Avant de commencer la partie, il faut créer le plateau de jeu avec les pièces déjà placées. Ceci est fait dans la fonction *init_game*()

```
/* créer un plateau de jeu avec les pions bien placés*/
void init_game(int plateau[]){

    //créer les cases vides
    for (int i = 0; i < TAILLE_PLATEAU; i++)
    {
        for (int j = 0; j < TAILLE_PLATEAU; j++)
        {
            plateau[i*TAILLE_PLATEAU + j] = CASE_VIDE;
        }
    }

    //créer les pions noires
    for(int i = 0; i < 4; i++)
    {
        for (int j = (i+1)%2; j < TAILLE_PLATEAU; j+=2)
        {
            plateau[i*TAILLE_PLATEAU + j] = PION_NOIR;
        }
```

```
    }

    //créer les pions blancs
    for(int i = TAILLE_PLATEAU-1; i > TAILLE_PLATEAU-5; i--)
    {
        for (int j = (i+1)%2; j < TAILLE_PLATEAU; j+=2)
        {
            plateau[i*TAILLE_PLATEAU + j] = PION_BLANC;
        }
    }
}
```

4.2.2 Jouer un coup

Cette partie n'a pas encore été implémentée en réseau mais a été réflechie et codée dans le fichier dame.c
Afin de jouer un coup, il faut tout d'abord choisir la pièce à déplacer. L'algorithme s'assure que le joueur sélectionne bien l'une de ses pièces

```
printf("chosir le pion à jouer.\n");
p = choisir_case();
printf("x = %d, y = %d\n", p.x, p.y);
while(plateau[p.x*TAILLE_PLATEAU + p.y]%2 != joueur || plateau[p.x*TAILLE_PLATEAU
{
    printf("\nVous devez choisir un pion qui vous appartient.\n");
    p = choisir_case();
    printf("x = %d, y = %d\n", p.x, p.y);
}
```

Ensuite, le joueur doit choisir la case destination. Comme pour le choix précédent, l'algorithme s'assure que la destination est bien une case vide.

```
printf("chosir la destination.\n");
p_dest = choisir_case();
while (plateau[p_dest.x*TAILLE_PLATEAU + p_dest.y] != CASE_VIDE)
{
    printf("choisir une destination valide\n");
    p_dest = choisir_case();
}
```

Finalement, une fonction est appelée pour faire d'autres vérifications comme la validité du coup (déplacement diagonal) ou dans le cas d'une prise, vérifier qu'il y a bien une pièce de l'adversaire à prendre. Cette fonction permet de changer la valeur d'une variable booléenne qui fait boucler l'algorithme jusqu'à ce que le coup soit enfin valide.

4.2.3 L'affichage

L'affichage est disponible uniquement dans le terminal pour le moment. Nous utilisons des boucles *for* afin d'afficher le contenu de notre tableau *plateau*[] en séparant chaque éléments par le caractère "|" afin de simuler les cases du plateau. Nous avons également rajouté des chiffres sur le coté afin de voir plus facilement à quelle ligne nous sommes. La même chose pour les colonnes arrive bientôt.

```
/* affiche l'état du plateau de jeu */
void afficher_plateau(int plateau[])
{
    for (int i = 0; i < TAILLE_PLATEAU; i++)
    {
        printf("%d  |", i);
        for (int j = 0; j < TAILLE_PLATEAU; j++)
        {
            printf("%d|", plateau[i*TAILLE_PLATEAU + j]);
        }
        printf("\n");
    }
    printf("\n");
}
```

4.2.4 Fin de la partie

Afin de savoir si la partie est finie, on a créer une fonction à appeler après chaque coup comportant une prise, qui vérifie que le nombre de pions blancs ou noirs n'est pas égal à 0. Si c'est le cas, la fonction retourne $TRUE$ et la partie est fini. Le gagnant est la dernière personne à avoir joué.

5 Améliorations possibles

Il est difficile de faire une partie sur des améliorations possibles quand le plus gros du travail reste à faire, mais c'est une partie qui nous tient à coeur car ce projet étant sur un dépot git, il nous sera très facile de le continuer sur notre temps libre.
Parmis ces améliorations, il y a celle du plateau de jeu. Etant composé à moitié de case blanches, et l'autre de case noires, et n'utilisant que ces dernières, il devrait être possible de réduire la taille stockée en mémoire. C'est une amélioration très légère, mais elle peut être très utile car nous envoyons le plateau de jeu assez souvent entre le serveur et les clients. Réduire la taille des paquets envoyés reste une priorité en réseau.
D'autre optimisations pourrait être les bienvenues sur les structures utilisées. Pas forcement au niveau du jeu en lui même (quoique...) mais plus au niveau

des structures pour le serveur et les clients. Cela permettrai peut être de pouvoir gérer les comptes correctement.

6 Conclusion

Bien que nous manquions cruellement de temps pour arriver à un stade qui nous satisfasse, ce projet nous as demandé de faire preuve d'initiative dans nos recherches et nous a permis de nous servir de nos connaissances acquises en cours afin de comprendre les notions de base du réseau. Le projet est disponible sur un dépôt git ce qui nous permettra de continuer ce projet hors du cadre de nos études. Nous aimerions rajouter au projet la possibilité de jouer seul contre "l'ordinateur" (en plus des fonctionnalités que nous n'avons pas encore implémentées).

7 Dépot git

`https://github.com/DragSoul/cDraughtsTCP`