



PROJET DE PROGRAMMATION UNIX

Réalisation d'un service de jeux :

Course dans un labyrinthe 2D

1. Objectif, Organisation et Évaluation

Objectif

Mise en œuvre et maîtrise approfondies des notions abordées dans l'UE Programmation Unix.

Nombre d'étudiants par groupe

Le projet est réalisé par groupe de deux ou trois étudiants.

Constitution des groupes

Les étudiants forment les groupes à leur convenance.

Déclaration des groupes

Un étudiant du groupe envoie, **au plus tard le 26/9/2018 16H00**, un mail à michel.soto@parisdescartes.fr ayant pour sujet : [PU] GROUPE PROJET et dans le corps la liste des membres du groupe avec pour chacun N°ETUDIANT NOM Prénom

Intégration dans la note finale

Le projet n'est pas obligatoire. Il est noté sur 20 et sa note s'ajoutera, sur la forme de 0 à 3 points de bonus, à la note de contrôle continu.

Les étudiants d'un même groupe peuvent avoir des notes différentes.

La note finale à l'UE de chaque étudiant sera :

$$\text{MAX} [((\text{Bonus_projet} + \text{DST}) + \text{examen})/2, \text{examen}]$$

Évaluation

L'évaluation sera faite à partir des livrables et d'une mini soutenance individuelle.

Dates

- 05/10/2018 - Livrable 1 : Conception de la base de données.
- 05/10/2018 - Livrable 2 : Implémentation de la base de données
- 30/11/2018 - Livrable 3 : code de `maze_cli` ainsi que de son `makefile`
- 30/11/2016 - Livrable 4 : code de `maze_ser` ainsi que de son `makefile`
- 30/11/2016 - Livrable 5 : fiche(s) rapport

2. Présentation

Le but de ce projet est la réalisation d'un service qui permet à 2 joueurs distants de jouer en utilisant Internet. Le jeu repose sur une partie *Serveur* et une partie *Client*. Pour jouer, chaque joueur devra posséder la partie client sur sa machine. Le serveur est capable de gérer plusieurs parties en même temps.

2.1. Le jeu

Le jeu consiste en une course entre 2 joueurs, A et B, dans un labyrinthe à deux dimensions. Le gagnant est, soit celui qui atteint le premier la sortie du labyrinthe, soit le joueur restant en cas d'abandon de son adversaire. Le labyrinthe ne possède qu'une seule entrée et qu'une seule sortie. Chaque joueur utilise un ordinateur relié au réseau au moyen d'une connexion TCP/IP. Sur l'écran de chaque joueur est visualisé le labyrinthe et la position courante des 2 joueurs. Chaque joueur utilise un caractère de jeux qui lui est propre pour matérialiser sa position dans le labyrinthe. Par exemple, joueur A utilise "@" et le joueur B utilise "#". Dès qu'un joueur a gagné, tout déplacement devient impossible. Au cours de la partie s'affiche le temps écoulé (en seconde) depuis le début de la partie.

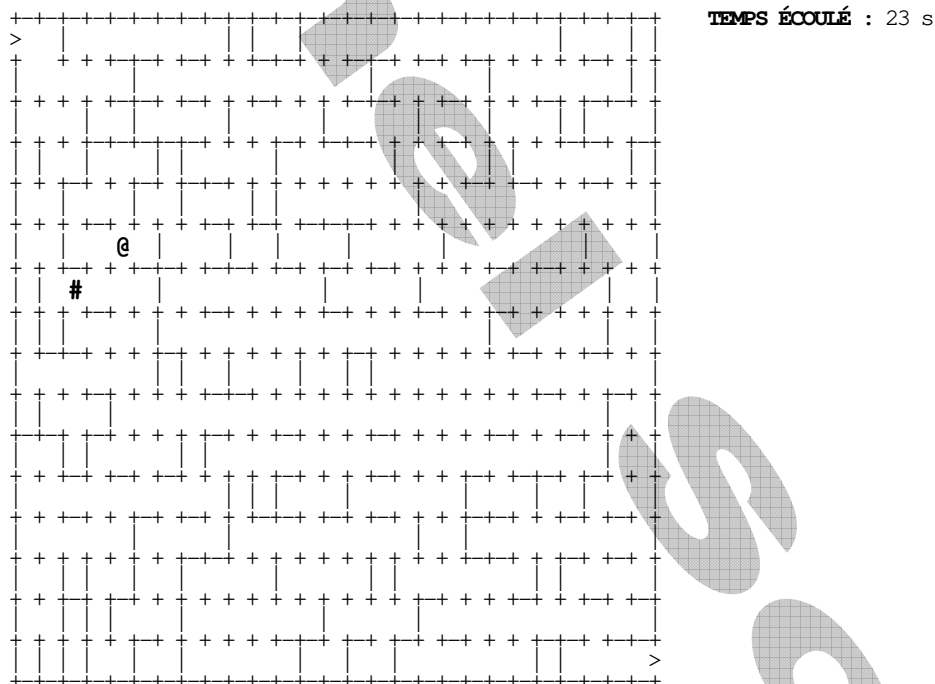


Figure 1 : Exemple de partie en cours.

Les joueurs pourront choisir le labyrinthe dans lequel ils vont s'affronter parmi un choix qu'il leur est proposé.

2.2. Règles de déplacement

- Chaque joueur contrôle son déplacement dans le labyrinthe avec les flèches ← (Gauche), → (Droite), ↑ (Haut) et ↓ (Bas) du clavier.
- Un joueur peut occuper une position déjà occupée par son adversaire

3. Le service du jeu

3.1. Fonctionnalités minimales du serveur

Le service doit permettre :

- D'enregistrer un joueur. Ce dernier devra choisir une chaîne de caractères (pseudo) qui l'identifie de manière unique et fournir son nom.
- D'identifier un joueur un joueur déjà enregistré à partir de son pseudo.
- De proposer une partie.
- De choisir un labyrinthe pour le joueur qui propose une partie.
- De connaître la liste des parties en attente de participant.
- De se joindre à une partie en attente de participant.
- De choisir un caractère de jeu pour ses déplacements
- D'abandonner à tout moment une partie.
- De connaître l'historique des parties **terminées** pour un labyrinthe donné ou un joueur donné.

En phase de jeux, le serveur envoie au client du joueur A les coordonnées de déplacement du joueur B et au client du joueur B les coordonnées de déplacement du joueur A.

3.2. Fonctionnalités supplémentaires du serveur

- Connaître la liste des parties en cours.
- Observer une partie en cours sans y participer

3.3. Fonctionnalités minimales du client

Le client sert d'interface entre le joueur et le service. Il propose au joueur un menu regroupant relatif aux fonctionnalités du service et il transmet au serveur les choix effectués par le joueur. Il lui permet, bien entendu de jouer.

En phase de jeu ou en phase d'observation simple, le client affiche l'état du labyrinthe après chaque déplacement par un des participants. Il envoie également au serveur les coordonnées de chaque déplacement de son joueur.

3.4. Fonctionnalités supplémentaires du client

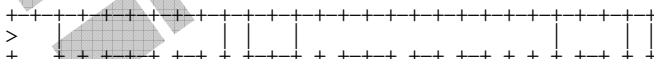
Les fonctionnalités supplémentaires du client découlent directement des fonctionnalités supplémentaires du service.

3.5. Gestion des joueurs, des parties et des labyrinthes

Cette gestion s'appuie sur une BD. Chaque joueur est enregistré dans la base. Un joueur est caractérisé par un pseudo unique, un nom et une date d'enregistrement dans la BD.

Chaque partie terminée est caractérisée par un n° unique, sa date, l'heure de début, sa durée en seconde et le nombre de déplacements effectués par le vainqueur. La BD doit permettre de connaître le gagnant et le perdant de chaque partie.

La topologie de chaque labyrinthe est décrite par plusieurs lignes (5 au minimum) composées des caractères '+', '-', ' ', '>' (entrée ou sortie) et '|'. Chaque ligne est caractérisée par un n° relatif au labyrinthe auquel elle appartient et par sa longueur. Par exemple, les ligne 1, 2 et 3 du labyrinthe de la Figure 1 sont les suivantes :



Il est précisé que :

- Un joueur ne participe au maximum qu'à une seule partie en cours.
- Un joueur ne peut être en attente que d'un adversaire au plus.

Un protocole de communication applicatif MTP (Maze Transport Protocol) régit les échanges entre le client et le serveur. Ce protocole repose sur plusieurs types de PDU (Protocol Data Unit). Chaque PDU est constituée d'un code suivi de zéro ou plusieurs champs selon ce code. Tous les champs ont une taille d'un octet sauf les champs qui contiennent une durée qui sont sur 2 octets, les champs qui contiennent un pseudo qui sont sur 8 octets et les champs qui contiennent une ligne de labyrinthe. Les PDU de type GET sont envoyées par le client au serveur. Les PDU de type PUSH sont envoyées par le serveur au client.

FORMATS DES PDU MTP

TYPE	CODE	CHAMP 1	CHAMP 2	CHAMP 3	CHAMP 4
GET CONNECT	05	Pseudo			
PUSH CONNECT	10	Connu/Inconnu	Taille message	Message	
GET DISCONNECT	15	Pseudo			
PUSH DISCONNECT	20	Taille message	Message		
GET INFO MAZES	25				
PUSH INFO MAZES	30	N° labyrinthe	Niveau	More/Last	
GET SELECTED MAZE	35	N° labyrinthe			
GET MAZE	40	N° labyrinthe			
PUSH MAZE	45	NB ligne	NB colonne		
PUSH LINE	50	Une ligne de labyrinthe			
GET REC	55	Pseudo			
PUSH REC	60	OK/Existe			
GET MOVE	65	X	Y		
PUSH MOVE	70	X	Y		
GET HIST MAZE	75	N° labyrinthe			
GET HIST PLAYER	80	Pseudo			
PUSH HIST	85	N° partie	NB déplacement du gagnant	Durée partie	More/Last
GET CHALLENGE	90	Pseudo	Caractère de jeu	N° labyrinthe	Niveau
GET MESSAGE	95	Pseudo A	Taille message	Message	
PUSH MESSAGE	100	Pseudo B	Taille message	Message	
GET ENDED GAME	105	N° partie			
PUSH ENDED GAME	110	Pseudo gagnant	Pseudo B	N° labyrinthe	
GET CHALLENGERS	115				
PUSH CHALLENGERS	120	Pseudo	N° Labyrinthe	Niveau	More/Last
GET JOIN	125	Pseudo B			
GET WINNER	130	Pseudo du gagnant	Durée partie		
PUSH WINNER	135	Pseudo du gagnant			
ERROR PROTOCOL	140	Code PDU			

TYPES DES PDU MTP COMMENTÉS

TYPE	COMMENTAIRE
GET CONNECT	Le joueur veut utiliser le service de jeu.
PUSH CONNECT	Si le pseudo est déjà enregistré, le champ 1 vaut connu (1) et inconnu (0) sinon. Transporte éventuellement un message du serveur qui doit être affiché par le client. Le nombre de caractères du message figure dans le champ 2
GET DISCONNECT	Le joueur ne veut plus utiliser le service
PUSH DISCONNECT	Le serveur accuse réception de la demande de déconnexion. Transporte éventuellement un message du serveur qui doit être affiché par le client. Le nombre de caractères du message figure dans le champ 1
GET INFO MAZES	Demande au serveur la liste des labyrinthes disponibles
PUSH INFO MAZES	Transporte le n° et le niveau d'un labyrinthe. Si ce labyrinthe n'est pas le dernier de la liste le champ 2 vaut More (1) et vaut Last (0) sinon.
GET SELECTED MAZE	Indique au serveur le n° du labyrinthe choisi pour la partie que le joueur propose
GET MAZE	Demande au serveur le tracé du labyrinthe dont le n° figure dans champ 1
PUSH MAZE	Transporte le nombre de ligne et de colonne du labyrinthe
PUSH LINE	Transporte une chaîne de caractères appartenant au labyrinthe demandé
GET REC	Demande l'enregistrement d'un nouveau joueur dont le pseudo est dans champ 1
PUSH REC	Si le pseudo du nouveau joueur n'est pas déjà pris par un autre joueur champ 1 vaut OK (1) et vaut existe (0) sinon
GET MOVE	Utilisé par le client pour informer le serveur d'un déplacement de son joueur en X (champ 1) et en Y (champ 2)
PUSH MOVE	Utilisé par le serveur pour informer un client d'un déplacement de son adversaire en X (champ 1) et en Y (champ 2)
GET HIST MAZE	Demande de l'historique des parties pour le labyrinthe dont le n° figure champ 1.
GET HIST PLAYER	Demande de l'historique des parties pour le joueur dont le pseudo figure champ 1.
PUSH HIST	Transporte le n° d'une partie, le nombre de déplacement du gagnant et la durée. Si cette partie n'est pas la dernière de la liste le champ 4 vaut More (1) et vaut Last (0) sinon.
GET CHALLENGE	Proposition d'une partie
GET MESSAGE	Le client envoie au serveur un message de son joueur à remettre du joueur A Le nombre de caractères du message figure dans le champ 2
PUSH MESSAGE	Le serveur envoie au client un message provenant du joueur B Le nombre de caractères du message figure dans le champ 2

GET ENDED GAME	Demande au serveur les informations d'une partie terminée dont le n° figure champ 1
PUSH ENDED GAME	Transporte les informations d'une partie terminée
GET CHALLENGERS	Demande la liste des parties en attente d'un adversaire
PUSH CHALLENGERS	Transporte les informations d'une partie en attente d'un adversaire. Si cette partie n'est pas la dernière de la liste le champ 4 vaut More (1) et vaut Last (0) sinon.
GET JOIN	Le joueur du client veut affronter le joueur B qui propose une partie.
GET WINNER	Un joueur vient d'atteindre la sortie du labyrinthe :il est vainqueur. Transporte la durée de la partie
PUSH WINNER	Annonce à un joueur que son adversaire a gagné.
ERROR PROTOCOL	Le client ou le serveur a reçu une PDU inconnue ou non appropriée dont le code est dans champ1.

4. Cahier des charges

4.1. Langage et bibliothèque de développement

Le projet sera développé en langage C avec la bibliothèque POSIX

Les entrée/sorties du client ne se feront pas avec la bibliothèque *stdio* mais avec la bibliothèque *ncurses*.

4.2. Utilisation de la Forge

Pour Chaque groupe, un projet sera créé sur la Forge à:

https://projets3.ens.math-info.univ-paris5.fr/projects/soto_projets_l3_progunix_2018_19

4.3. Structure du service

- Le client d'un joueur sera implémenté dans un programme dont l'appel sera :
`maze_cli nom_serveur port_serveur`
où *nom_serveur* est le nom de la machine qui héberge le serveur du jeu et *port_serveur* est le port du serveur sur sa machine.
- Le serveur sera implémenté dans un programme dont l'appel sera :
`maze_ser port_serveur`
où *port_serveur* est le port du serveur sur sa machine.

5. Travail demandé

Vous devez réaliser en langage C les fonctionnalités minimales du serveur et du client tels que décrits précédemment. Vous êtes libre de réaliser ou non une ou plusieurs fonctionnalités supplémentaires. Il en sera tenu compte dans votre note de projet. Il vous est demandé de vous inspirer de la démarche ci-dessous pour réaliser votre travail.

5.1. Livrables

5.1.1. Le 05/102018 - Livrable 1 : Conception de la base de données

- Schéma conceptuel selon le modèle entité/association pour ce jeu.
- Schéma relationnel de la base de données.

Dépôt

Ce livrable sera déposé sur la Forge sur l'onglet *Documents* de votre projet dans un fichier nommé `MDCR_PPUn` au format pdf où `n` est le n° de votre groupe de projet.

5.1.2. Le 05/10/2018 - Livrable 2 : Implémentation de la base de données

- Implémentation du schéma relationnel en PostgreSQL en utilisant la BD *votre login*.
- Initialiser le contenu de cette BD avec les labyrinthes fournis.

Dépôt

Ce livrable sera déposé sur la Forge sur l'onglet *Fichiers* de votre projet dans un fichier texte nommé `SQL_PPUn.sql` où `n` est le n° de votre groupe de projet.

5.1.3. Le 30/11/2018 - Livrable 3 : code de `maze_cli` ainsi que de son `makefile`

Dépôt

Ce livrable sera déposé sur la Forge sur l'onglet *Fichiers* de votre projet dans un fichier texte nommé `maze_cli_PPUn` où `n` est le n° de votre groupe de projet.

5.1.4. Le 30/11/2016 - Livrable 4 : code de `maze_ser` ainsi que de son `makefile`

Dépôt

Ce livrable sera déposé sur la Forge sur l'onglet *Fichiers* de votre projet dans un fichier texte nommé `maze_ser_PPUn` où `n` est le n° de votre groupe de projet.

5.1.5. Le 30/11/2016 - Livrable 5 : fiche(s) rapport

Sur une ou plusieurs feuilles A4, vous écrirez tout ce que vous jugez nécessaire à l'évaluation de votre travail.

Dépôt

Ce livrable sera déposé sur la Forge sur l'onglet *Documents* de votre projet dans un dossier compressé nommé `Liv5_PPUn` au format zip où `n` est le n° de votre groupe de projet.

ANNEXE TECHNIQUE

Gestion des entrées/sorties

Les entrées/sorties du client ne se feront pas avec *stdio* mais avec *ncurses*. Les principales caractéristiques de la librairie *ncurses* sont :

- Définition d'un système de coordonnées (x, y) sur l'écran. L'origine de ce système a pour coordonnées (0,0) et se situe en haut à gauche de l'écran.
- Positionnement du curseur d'affichage n'importe où sur l'écran grâce au système de coordonnées.
- Suppression de l'écho écran des caractères frappés par l'utilisateur.
- Validation immédiate d'un caractère sans utiliser la touche *entrée* (retour chariot).

Squelette d'un programme utilisant les *ncurses* :

```
#include <ncurses.h>
int main(void)
{
    initscr(); /* Première fonction exécutée pour initialiser un programme utilisant ncurses */
    ... ..

    endwin(); /* Cette fonction doit être appelée avant de quitter le programme en mode curses. */
              /* Elle permet de rétablir l'ancien mode du terminal. */
    return 0;
}
```

Principales fonctions des *ncurses* :

Initialisation et fin d'exécution

- *initscr()*: C'est la première fonction exécutée pour initialiser un programme utilisant *ncurses*.
- *endwin()*: Cette fonction doit être appelée avant de quitter le programme en mode curses. Elle permet de rétablir l'ancien mode du terminal.
- *cbreak()*, *nocbreak()*: Normalement l'entrée d'un caractère doit être validée par l'appui sur la touche *entrée* ou le caractère saut de ligne. La fonction *cbreak()* rend immédiatement disponible le caractère saisi. *nocbreak* rétablit le mode par défaut.
- *noecho()*, *echo()*: La fonction *noecho()* désactive l'affichage sur l'écran d'une touche frappée au clavier. La fonction *echo()* rétablit l'affichage du caractère saisi.

Sortie écran

- *int clear()*, *int erase()* : Fonctions d'effacement d'écran. La fonction *erase()* place un caractère blanc à chaque emplacement d'écran. La fonction *erase()* réalise le même travail, mais appelle la fonction *clearok* qui lancera automatiquement un nouveau nettoyage d'écran à la suite du prochain appel à *refresh()*.
- *int clrtoebot()* : Cette fonction efface l'écran à partir de la position du curseur.
- *int clrtoeol()* : Cette fonction efface l'écran à partir de la position du curseur jusqu'à la fin de la ligne.
- *int refresh()* : Fondamental : toutes les modifications effectuées sur une fenêtre sont réalisées en mémoire. Pour que la modification soit effective sur écran, il faut utiliser la fonction *refresh()*.

- `int printf(char * format, ...)` : La syntaxe est tout à fait similaire à celle de `printf`. Elle permet d'afficher une chaîne de caractères à la position du curseur. Les options de formatage sont les mêmes que pour la fonction `printf`
- `int move(int y, int x)` : Le curseur est placé en `x, y`. Attention l'origine de la fenêtre (coin supérieur gauche) est `0,0`. Cette modification est réalisée dans la fenêtre virtuelle en mémoire; il faut utiliser `refresh()` pour la rendre effective sur l'écran physique.
- `int mvaddchstr(int y, int x, const chtype *chstr)` : Cette fonction affiche la chaîne `chstr` à la position `y, x` de la fenêtre. Attention `refresh` est toujours nécessaire.

Entrée clavier

- `getch()` : Cette fonction permet la saisie d'un caractère.
- `int scanw(char * format, ...)` : Fonction semblable à `scanf`.

Un exemple

Faites marcher et modifier le programme suivant pour vous exercer avec les ncurses :

```
/* ===== */
/* Programme de familiarisation avec les ncurses. */
/* Il permet de déplacer le curseur d'affichage et en */
/* utilisant les flèches du clavier et d'afficher les */
/* coordonnées courantes du curseur */
/* COMPILATION: cc test_ncurses.c -o test_ncurses -lncurses */
/* ===== */
#include <ncurses.h>
#define DELAI 15000
#define XMAX 25
#define YMAX 80
int x; // axe vertical bas
int y; // axe horizontal droite
int x,y,k;
int c;

int main(int argc, char *argv[])
{
    initscr();
    erase();
    keypad(stdscr, TRUE); /* enable keyboard mapping */
    (void) nonl(); /* tell curses not to do NL->CR/NL on output */
    (void) cbreak(); /* take input chars one at a time, no wait for \n */
    (void) noecho(); /* don't echo input */

    x=0;
    y=0;

    move(x,y);

    printf("");
    refresh();
    for (c = getch(); c!='q' && c!='Q'; c = getch()) {

        switch (c) {
            case KEY_LEFT : /* Traitement flèche gauche
                if (y>0) {
                    y--;
                    erase();
                    move(x,y);
                    printf("X=%d,Y=%d",x,y);
                    refresh();
                }
            }
        }
    }
}
```

```

    }
    break;

    case KEY_RIGHT : // flèche droite
        if (y<YMAX){
            y++;
            erase();
            move(x,y);
            printw("X=%d,Y=%d",x,y);
            refresh();
        }
        break;
    case KEY_DOWN : // flèche bas
        if (x<XMAX){
            x++;
            erase();
            move(x,y);
            printw("X=%d,Y=%d",x,y);
            refresh();
        }
        break;
    case KEY_UP : // flèche haut
        if (x>0){
            x--;
            erase();
            move(x,y);
            printw("X=%d,Y=%d",x,y);
            refresh();
        }
        break;
    case 'q' : // q pour sortir
    case 'Q' :
        break;
    default :
    } // switch (c)
} //for

```

```

erase();
refresh();
endwin();
return 0;

```

```

}

```