

Programmation Concurrente

Réactive et Répartie

Devoir de Programmation : Des Robots qui Ricochent

Morvan Lassauzay
morvan.lassauzay@etu.upmc.fr

Victor Nea
victor.nea@etu.upmc.fr

Encadrants :
Emmanuel Chailloux, Romain Demangeon, Philippe Esling, Tong Lieu

Introduction

Le problème consiste en la réalisation d'une adaptation du jeu de *Rasende Roboter* au travers d'une application clients-serveur. Le jeu consiste à déplacer des robots selon le principe de « Sokoban » sur un plateau de jeu de 16x16 cases comportant des murs, afin d'acheminer le robot voulu jusqu'à la case cible, le but du jeu étant de proposer une solution comportant le moins de déplacements possible. Pour cela un système de phase est mis en place, avec une phase de réflexion, puis une phase d'enchère et enfin une phase de résolution dans laquelle les joueurs proposent leur solution.

Pour résoudre le problème posé, nous avons eu recours aux langages de programmation Java pour la réalisation du serveur et C pour le client. La bibliothèque Ncurses a quant à elle été utilisée pour élaborer l'interface utilisateur dans le programme client.

Le programme client

Architecture générale

Le programme client est composé de 3 duo de fichiers .c .h. Les fichiers « **interfaces** » décrivent toutes les méthodes permettant la manipulation des fenêtres Ncurses ainsi que l'écriture à l'intérieur de celles-ci. Ce sont des fichiers réutilisables qui ne tiennent donc pas compte du programme dans lequel ils sont utilisés.

Les fichiers « **plateau** » regroupent les différentes structures utilisées pour représenter le jeu, soit, le plateau, les robots, la cible et les cases. Ils contiennent également toutes les méthodes permettant d'initialiser et d'afficher ces différentes structures.

Pour finir les fichiers « **client** » englobent quant à eux l'ensemble des méthodes permettant de gérer le programme en lui-même. On y trouve donc les méthodes de gestion de la connexion au serveur, les méthodes de gestion des différentes phases, et les méthodes d'affichage des différents éléments d'interaction et d'information pour le joueur. En regardant de plus près ces méthodes on peut voir que les différents éléments cités précédemment s'y intersectent fortement. Il a donc été choisi de laisser ces éléments dans le même fichier. A noter que la méthode « **main** » est également présente en fin du fichier « **client.c** ».

Le jeu : représentation et fonctionnement

Pour la représentation du plateau de jeu et de ses éléments, 4 structures très simples ont été utilisées:

- Les **robots** et la **cible** sont composé de deux entiers permettant d'indiquer sur quelle case ils se trouvent, par exemple la case de la colonne 2 et de la ligne 3.
- Une **case** est constituée de deux entiers correspondant respectivement aux positions (en pixel Ncurses) x et y de la case dans la fenêtre d'affichage du plateau. De plus une case contient 6 entiers faisant office de booléens pour indiquer si la case comporte un robot, un mur à sa gauche, à sa droite, en haut, en bas, ou bien encore si elle est la case cible.
- La structure du **plateau** comprend donc quant à elle un matrice de cases, les 4 robots du jeu, et la cible.

Des macros ont été utilisées pour permettre de redéfinir très simplement la taille du plateau de jeu et des cases. Il est également rapide de rajouter ou retirer des robots, néanmoins certaines méthodes devront tout de même être modifiées puisqu'il est souvent nécessaire de traiter un robot de façon distincte étant donnée que chaque robot dispose de sa propre couleur.

Le programme client ne dispose pas d'un réel contrôle sur le jeu. Son fonctionnement consiste à suivre les informations qu'il reçoit du serveur. Pour recevoir ces informations il utilise un thread nommé « **lecteur** » qui va ensuite traiter les informations reçues selon les règles du jeu. L'algorithme de réception des commandes du serveur est le suivant :

Variables :

messRecv : chaîne

c : caractère

cpt : entier

Début :

cpt ← 0

TANT QUE vrai FAIRE

 TANT QUE c est différent de '\n' FAIRE

 c ← prochain caractère envoyé par le serveur

 messRecv[cpt] ← c

 cpt ← cpt + 1

 FIN TANT QUE

 messRecv[cpt] ← '\0'

 cpt ← 0

 c ← '\0'

 traitement de la commande reçu

FIN TANT QUE

Fin

Le traitement de la commande reçu peut être vu comme le simple fait d'afficher les informations reçues à l'écran, cela pouvant nécessiter l'initialisation des différentes structures présentées précédemment.

L'interface

Nous avons donc vu que le rôle du client était donc de permettre au joueur de visualiser l'état du jeu et d'interagir avec celui-ci. Pour cela le client a recours à une structure de **fenêtre** encapsulant deux fenêtres Ncurses. La première fenêtre Ncurses permet de définir un contour pour la fenêtre qui sera affichée à l'écran. La deuxième est quant à elle la fenêtre dans laquelle les éléments à afficher seront réellement présents. Bien entendu notre structure de fenêtre dispose également de quatre entiers pour définir sa position, sa largeur et sa hauteur.

Pour réussir à afficher toutes les informations nécessaires à l'écran le programme client utilise plusieurs fenêtres :

- Une fenêtre pour afficher le plateau de jeu.
- Une fenêtre pour afficher l'énigme du tour en cours.
- Une fenêtre pour indiquer les différentes informations concernant le déroulement de la partie.

- Une fenêtre permettant d'interagir avec le joueur.

À l'exception des fenêtres, Ncurses ne fonctionne pas à l'aide d'objet graphique. Elle n'est en fait qu'une bibliothèque permettant de simplifier l'affichage des caractères (un pixel par caractère). Pour réaliser un affichage plusieurs méthodes doivent donc être utilisées. Dans un premier temps il est souvent nécessaire de se placer à l'endroit auquel on souhaite afficher quelque chose. Pour cela on utilise la méthode « `fenetre_setpos` ». Ensuite on peut afficher un caractère en couleur à l'aide de la méthode « `fenetre_affcar` », ou afficher un message entier à l'aide de la méthode « `fenetre_affch` ».

A un instant donné le thread lecteur et le programme principal risque de devoir afficher en

même temps des informations différentes. Pour éviter le mélange des éléments à afficher un mutex est utilisé afin de protéger l'accès aux fenêtres.

Ncurses permet également de récupérer les inputs de l'utilisateur à un endroit précis, soit par une entrée au clavier, soit par un clic de souris. Ces entrées vont être dans notre cas récupérées dans la fenêtre dite de sélection. Une variable « **phase** » indiquant la phase dans laquelle la partie se trouve, permet de savoir quel input autoriser à un instant donné. Elle permet aussi d'afficher le curseur au bon endroit au bon moment.

Remarque : Une animation à voulu être proposer pour visualiser les solutions proposées mais n'a pas pu être implanter sans engendrer diverses problèmes. Néanmoins pour disposer d'un aperçu il suffit de dé-commenter la ligne 483 du fichier client.c. L'animation apparaît alors chez les clients qui n'ont pas proposés la solution.

Le serveur

Le serveur de l'application a été développé en Java.

Le projet contient deux principaux dossiers :

- **src**
- **res**

Le dossier **src** contient tout le code source de l'application et le dossier **res** contient les ressources à savoir les fichiers textes pour les plateaux et les énigmes du jeu.

Dossier src

Celui-ci est composé de différents dossiers (des packages en réalité).

Les différents packages sont les suivants :

- **app**
- **enumeration**
- **exception**
- **rasenderroboter**
- **server**
- **util**

Le package **app** :

- **Main** : point d'entrée du programme et qui lance tout simplement le serveur.

Le package **enumeration** :

- **Color** : définit les couleurs des robots.
- **CommandCS** : définit les commandes du client vers le serveur (par exemple CONNEXION/).
- **CommandSC** : définit les commandes du serveur vers le client.
- **Direction** : définit les directions pour le déplacement des robots.
- **Phase** : définit les différentes phases de jeu.
- **Wall** : définit les murs pour chaque case.

Le package **exception** :

- **InvalidShiftingRobotException** : exception qui permet d'indiquer qu'un déplacement d'un

robot est invalide (par exemple la commande RV est invalide).

Le package **rasenderroboter** (le plus important) :

- **Board** : définit tout simplement le plateau de jeu par le biais d'un tableau 2D de cases (classe **Tile**).
- **Tile** : définit une case d'un plateau. Une case peut avoir des murs sur chacun de ses côtés. Elle peut également contenir un robot et/ou la cible.
- **Robot** : définit un pion robot.
- **Target** : définit un pion cible.
- **ShiftingRobot** : définit un seul déplacement d'un robot à l'aide d'une couleur et d'une direction (par exemple le déplacement RH).
- **Player** : définit la classe d'un joueur. A chaque fois qu'un joueur se connecte au serveur, ce joueur est ajouté au jeu (via la classe **Game**). Cette classe « implements » l'interface **Runnable** pour que chaque client puisse envoyer des commandes au serveur de façon parallèle. Chaque instance de la classe **Player** est donc lancée dans un **Thread** et elle contient une instance vers le moteur de jeu (classe **Game**). Cette classe s'occupe donc de recevoir les commandes pour chaque client et en fonction de la commande reçue, l'instance Player va appeler une méthode de la classe **Game** pour traiter la commande.
- **Game** : c'est le moteur de jeu. Elle « implements » l'interface **Runnable**. En effet, comme c'est le **ThreadPooledServer** qui instancie le jeu, il faut que le **ThreadPooledServer** puisse continuer à recevoir des connexions mais qu'il exécute également le jeu en parallèle. La méthode s'occupe d'exécuter la boucle principale qui va tourner en boucle pour gérer les différents Timer pour les différentes phases (voir les méthodes **waitEndOfXXXPhase**, où XXX représente la phase actuelle). En parallèle, les commandes envoyées par les clients sont traitées par les méthodes **manageXXXPhase**, où XXX représente la phase actuelle.

Le package **server** :

- **ThreadPooledServer** : c'est la classe serveur qui contient une **ThreadPool**. Chaque fois qu'un client se connecte, un objet **Player** est créé et il est exécuté dans la **ThreadPool**. Le port utilisé par défaut est le port 2016.

Le package **util** :

- **CommandParser** : contient la méthode **getCommands** qui permet de parser une commande reçue. Si l'on reçoit par exemple la commande « CONNEXION/user », **getCommands** retournera la liste suivante : [CONNEXION, user].
- **Logger** : permet de logger de message au sein du programme.

Dossier res

Le dossier **boards** :

Les différents plateaux sont stockés en dur dans des fichiers textes. La première ligne de chaque fichier texte définit le score objectif du plateau. Les plateaux sont nommés suivant ce pattern :

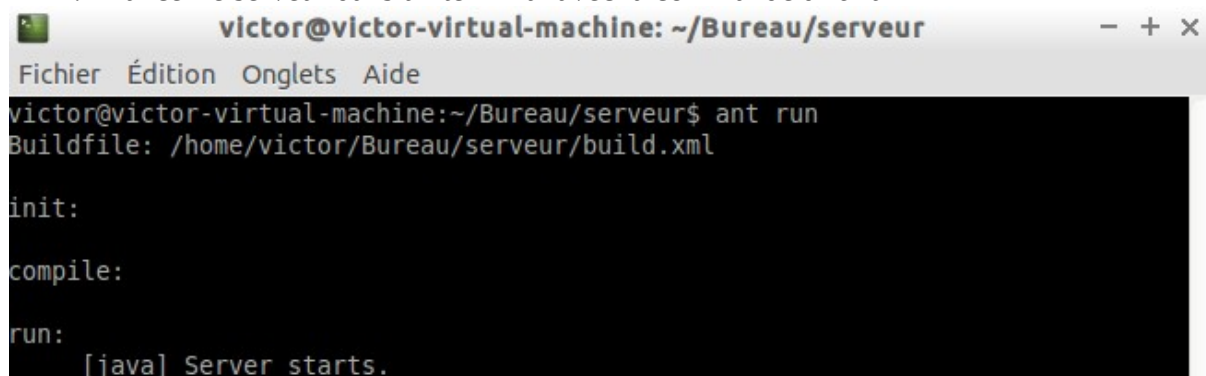
boardx.txt , où $x \in \mathbb{N}$.

Le dossier **enigmas** :

Il contient autant de dossier qu'il y a de plateaux. Par exemple si on a les plateaux **board1.txt** et **board2.txt**, on aura les dossiers **board1** et **board2**. Chaque sous dossier **boardx** contient les énigmes du plateau correspondant. Ils sont nommés suivant le pattern **enigmex.txt**, où $x \in \mathbb{N}$. Chaque fichier **enigmex.txt** contient ces informations : la première ligne représente le robot qui doit atteindre la cible, les 4 autres lignes suivantes représentent les robots et leur position (x, y) et la dernière ligne représente la position de la cible (x, y).

Manuel d'utilisation

1. Lancer le serveur dans un terminal avec la commande **ant run**



```
victor@victor-virtual-machine: ~/Bureau/serveur
Fichier Édition Onglets Aide
victor@victor-virtual-machine:~/Bureau/serveur$ ant run
Buildfile: /home/victor/Bureau/serveur/build.xml

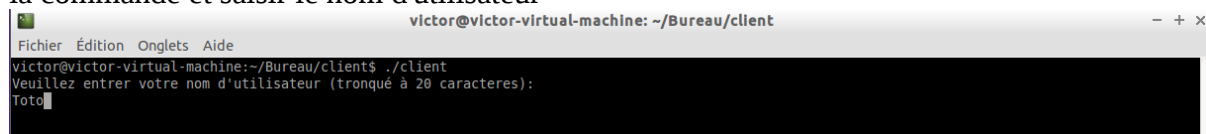
init:

compile:

run:
[java] Server starts.
```

Remarque : Pour changer l'adresse du serveur, il faut modifier le fichier **build.xml** et changer l'adresse à la ligne 23. Il est par défaut défini à l'adresse **localhost**. Vu qu'il a été spécifié qu'il fallait utiliser le port 2016 dans le sujet, le port a été spécifié directement dans le code. Cependant on aurait bien pu également le spécifier dans le fichier **build.xml**.

2. Lancer un ou plusieurs clients avec la commande **./client** dans différents terminaux avec la commande et saisir le nom d'utilisateur



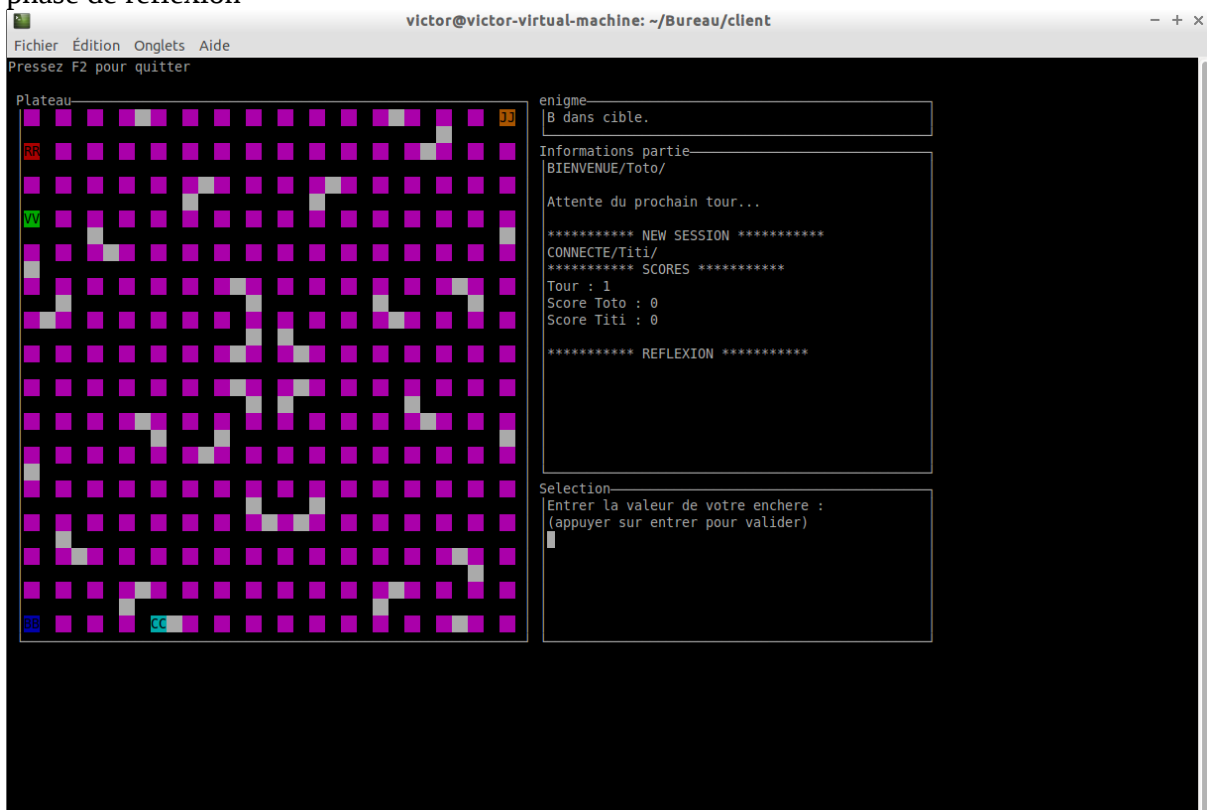
```
victor@victor-virtual-machine: ~/Bureau/client
Fichier Édition Onglets Aide
victor@victor-virtual-machine:~/Bureau/client$ ./client
Veuillez entrer votre nom d'utilisateur (tronqué à 20 caracteres):
Toto
```

Remarque : Lors de la première utilisation il est nécessaire d'exécuter la commande **make** au préalable.

3. Une fois le client lancé, on attend que d'autres joueurs se connectent pour démarrer la session



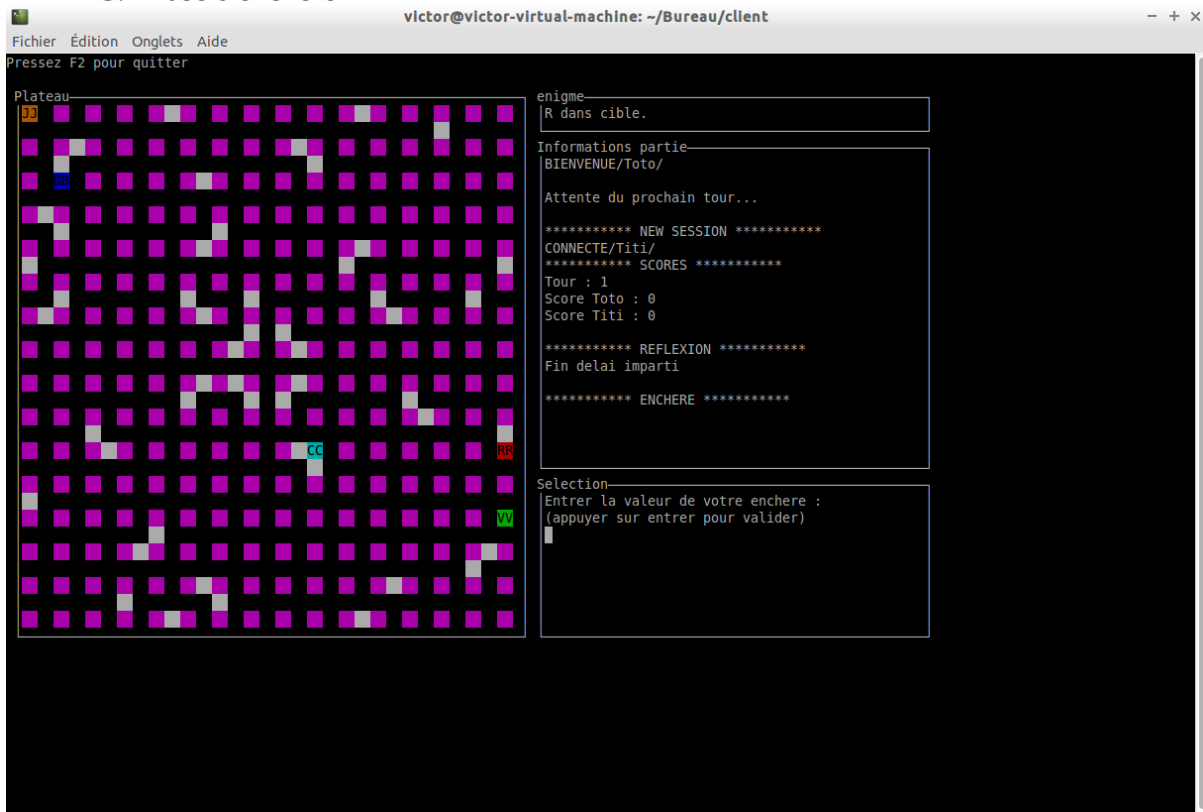
4. Une fois qu'il y a au moins deux joueurs, le jeu peu démarrer : on passe alors dans la phase de réflexion



RR représente le robot rouge, JJ le robot jaune, BB le robot rouge, VV le robot vert et CC la cible. On peut voir alors que l'énigme s'affiche dans le cadre **enigme**. On peut désormais saisir une enchère dans le cadre **Selection**. Il suffit ici d'entrer une enchère au clavier puis d'appuyer sur la

touche **Entrée**.

5. Phase d'enchère



Tout comme la phase de réflexion, il suffit de saisir une enchère au clavier puis d'appuyer sur la touche **Entrée**.

6. Phase de résolution



Ici, il faut d'abord cliquer avec la souris sur une des couleurs (qui représentent les robots) R, B, J ou V puis cliquer sur une des directions H, B, D ou G dans le cadre de **Selection**. Il est bien sûr possible de saisir plusieurs couples robots/directions. Le bouton **Correction** permet d'effacer la toute dernière saisie. Une fois le choix effectuer, il suffit d'appuyer sur le bouton **Envoyer** pour confirmer la solution.

Remarque: Différents plateaux et énigmes sont disponibles. Une solution triviale est toujours proposée afin de simplifier les tests.

7. A tout moment du jeu, il suffit d'appuyer sur la touche **F2** pour se déconnecter.

