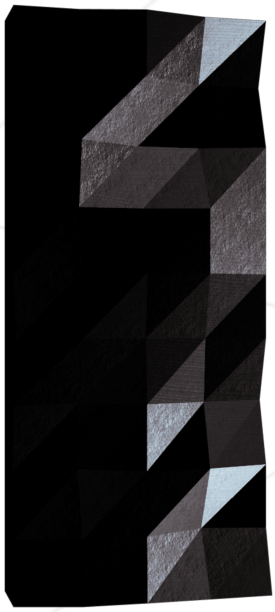


zappy

Projet UNIX cool

Résumé: Ce projet consiste à implémenter un jeu multi-joueurs en réseau TCP/IP

Table des matières

Chapitre I

Préambule

Zappy est un jeu entièrement automatique où des programmes informatiques jouent entre eux. Le jeu se déroule selon une vitesse définie par une unité de temps. Chaque action dans le jeu prend une durée proportionnelle à cette unité de temps.

L'unité de temps est définie par la fonction 1/t :

- `t` est passé en paramètre au démarrage du serveur,
- une unité de temps dure `1/t secondes`

Voici les différentes parties constituant ce jeu :

- un serveur qui contient le terrain, les ressources, et gère la logique et le timing du jeu.
- un ou plusieurs clients qui se connectent au serveur et "pilotent" chacun un joueur. Les joueurs sont répartit en équipe.
- un client graphique qui se connecte au serveur et affiche le terrain et ce qu'il s'y passe.

Chapitre II

Présentation du jeu

II.1 Géographie

Le jeu consiste à gérer un monde et ses habitants. Ce monde, nommé Trantor est géographiquement constitué de plaines ne comprenant aucun relief : ni cratère, ni vallée, ni montagne. Le plateau de jeu représente la totalité de la surface de ce monde comme un planisphère. Si un joueur sort par la droite du plateau, il rentrera par la gauche.

Le jeu se joue par équipe. L'équipe gagnante est celle dont six joueurs auront atteint l'élévation maximale.

II.2 Ressources

Le milieu dans lequel on évolue est plutôt riche en ressources, tant minières qu'alimentaires. Ainsi, il suffit de déambuler sur cette planète pour découvrir cette succulente nourriture ou une multitude de pierres de diverses natures.

Ces pierres sont de six genres distincts. Il existe la linemate, la deraumère, le sibur, la mendiane, le phiras, la thystame. La génération des ressources est réalisé par le serveur. Cette génération doit être aléatoire et comprenant des règles de logique.

II.3 Activités

Les habitants de Trantor vaquent à deux types d'occupation :

- récupérer de la nourriture pour se nourrir et ne pas mourrir de faim,
- rechercher et amasser des pierres pour fabriquer des totems, procéder à un rituel d'élévation et passer au niveau supérieur.

L'élévation représente une activité importante du Trantorien.

II.4 Individus

L'habitant est pacifique. Il n'est ni violent ni agressif.

Il déambule gaiement en quête de pierres et s'alimente au passage.

Il croise sans difficulté ses congénaires sur la même unité de terrain et voit aussi loin que ses capacités visuelles le lui permettent.

Le Trantorien est immatériel, il est flou et occupe toute la case dans laquel il se trouve. Il est impossible de distinguer son orientation lorsqu'on le croise.

La nourriture que le trantorien ramasse est la seul ressource dont il a besoin pour vivre. Une unité de nourriture permet de survivre 126 unité de temps, donc 126/t secondes.

II.5 Rituel d'élévation

L'objectif de tous est de s'élever dans la hiérarchie Trantorienne. Ce rituel qui permet d'accroître ses capacités physiques et mentales doit être effectué selon un rite particulier. Il faut en effet réunir sur la même unité de terrain :

- une combinaison de pierres,
- un certain nombre de joueurs de même niveau.

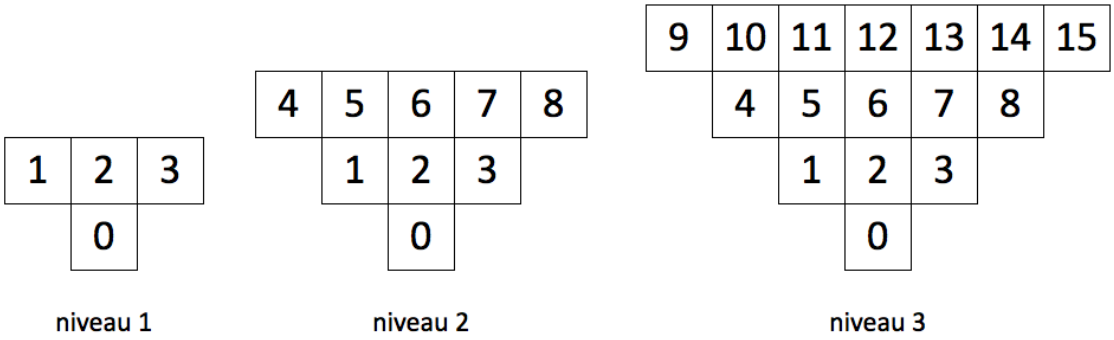
Un joueur débute l'incantation et l'élévation est alors en cours. Il n'est pas nécessaire que les joueurs soient de la même équipe. Seul leur niveau importe. Tous les joueurs du groupe réalisant l'incantation atteignent le niveau supérieur.

Transmis de génération en génération, le secret de l'élévation se résume ainsi :

élévation	Est requis						
niveau	nombre de joueurs	linemate	deraumere	sibur	mendiane	phiras	thystame
1-2	1	1	0	0	0	0	0
2-3	2	1	1	1	0	0	0
3-4	2	2	0	1	0	2	0
4-5	4	1	1	2	0	1	0
5-6	4	1	2	1	3	0	0
6-7	6	1	2	3	0	1	0
7-8	6	2	2	2	2	2	1

II.6 Vision

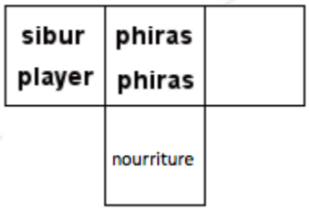
Pour diverses raisons, le champ de vision des joueurs est limité. A chaque élévation, la vision augmente d'une unité de mesure en avant et d'une de chaque côté de la nouvelle rangée. Ainsi on obtient, par exemple, pour les 3 premiers niveaux et par conséquent les deux premières élévations les visions suivantes (notre joueur occupe la case 0) :



Pour que le joueur ait connaissance de son entourage, le client envoie la commande 'VOIR', et le serveur répond la chaîne de caractères suivante (schéma pour un niveau 1) :

```
{contenu-case-0, contenu-case-1, contenu-case-2, contenu-case-3}
```

Notre joueur ne se voit pas lui-même, de plus lorsque sur une case se trouve plus d'un objet, ils sont tous indiqués et séparés par un espace :



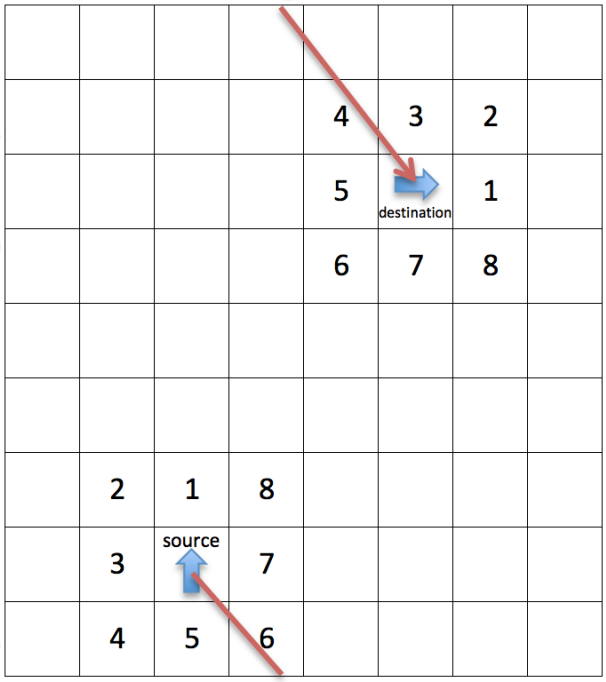

```
{nourriture, player sibur, phiras phiras, }
```

II.7 Transmission du son

Le son est une onde qui se propage de manière linéaire.

Tout les joueurs entendent les broadcasts sans savoir qui les émet. Ils percoivent uniquement la direction d'où provient le son et le message émis. La direction est indiquée par le numéro de la case franchie par le son avant d'arriver dans la case du joueur. Cette numérotation est effectuée par l'attribution du '1' à la case devant le joueur, puis à un décomptage des cases entourant le joueur dans le sens trigonométrique (anti-horaire). Le monde étant sphérique le trajet que l'on choisira pour le son sera le plus court chemin reliant l'émetteur au joueur pour lequel on le calcule.

L'exemple suivant indique le trajet du son que l'on doit choisir, ainsi que la numérotation des cases autour du joueur. Le joueur reçoit le broadcast par la case 3.



Dans le cas où le broadcast est émis à partir de la même case que le joueur récepteur, il recevra le message provenant de la case 0.

Chapitre III

Le travail

III.1 Le serveur

Un serveur réalisé en C sur le dump de l'école aura comme tâche de gérer le monde et ses habitants.

```
Usage: ./serveur -p <port> -x <width> -y <height> -n <team> [<team>] [<team>] ... -c <nb> -t <t>
  -p numero de port
  -x largeur du Monde
  -y hauteur du Monde
  -n nom\_equipe\_1 nom\_\_equipe_2 ...
  -c nombre de client autorises au commencement du jeu
  -t diviseur de l'unite de temps (plus t est grand, plus le jeu va vite)
```

Le serveur s'exécute sous la forme d'un seul process et un seul thread (ce qui veut dire : PAS DE THREAD !)

III.2 Le client

Un client réalisé en C, en PHP, en Perl, en pyhton, etc... (tant que ça fonctionne sur les dumps), pilotera un habitant par des ordres qu'il enverra au serveur.

```
Usage: ./client -n <team> -p <port> [-h <hostname>]
  -n nom d'equipe
  -p port
  -h nom de la machine par defaut c'est le localhost
```

Le client est autonome, après son lancement l'utilisateur n'influe plus sur son fonctionnement. Il pilote un drone (joueur) comme dans le projet corewar.

III.3 Les équipes

Au commencement une équipe est composée de n joueurs et seulement n. Chaque joueur est piloté par un client. Les clients ne peuvent pas communiquer ou échanger entre eux des données en dehors du jeu, de quelque façon que ce soit.

Au début le client possède 10 unité de vie, il peut donc survivre 1260 unités de temps, soit 1260/t secondes.

III.4 Les commandes

Chaque joueur répond aux actions suivantes et uniquement avec la sytaxe suivante :

Action	**Commande**	**Delai**	**Reponse**
avance d'une case	`avance`	7/t	`ok`
tourne à droite de 90 degrés	`droite`	7/t	`ok`
tourne à gauche de 90 degrés	`gauche`	7/t	`ok`
voir	`voir`	7/t	`{case1, case2, ...}`
inventaire	`inventaire`	1/t	`{phiras n, sibur n, ...}`
prendre un objet	`prend <objet>`	7/t	`ok/ko`
poser un objet	`pose <objet>`	7/t	`ok/ko`
expulse les joueurs de la case	`expulse`	7/t	`ok/ko`
broadcaster	`broadcast <texte>`	7/t	`ok`
débuter l'incantation	`incantation`	300/t	`elevation en cours`
			`niveau actuel : K`
forker un joueur	`fork`	42/t	`ok`
connaitre le nb de connections non utilisée par l'équipe	`connect_nbr`	0/t	valeur
mort d'un joueur	-	-	`mort`

Toutes les commandes sont transmises via une chaîne de caractères terminée par un retour à la ligne.

III.5 Dialogue client/serveur

Le dialogue entre le client et le serveur s'effectuera via sockets et tcp. Le port utilisé sera indiqué en paramètre des programmes.

Le client envoie ses requêtes sans attendre leur réalisation, le serveur renvoie un message confirmant le bon déroulement de l'exécution des requêtes.

La connection du client au serveur se déroule comme suit :

le client ouvre une socket sur le port (transmis en paramètre) du serveur ; puis le serveur et le client dialogue comme suit :

Message du client	Message du serveur
	`BIENVENUE\n`
<nom-equipe>\n	
	<nb-client>\n
	<x> <y>\n

Le `nb-client` indique le nombre de clients pouvant encore être acceptés par le serveur pour l'équipe `nom-equipe`.

Si ce nombre est supérieur à 1 un nouveau client se connecte.

`Le client peut envoyer jusqu'à 10 requêtes successives sans avoir de réponse du serveur. Au delà de 10 le serveur ne les prendra plus en compte.`

Le serveur exécute les requêtes des clients dans l'ordre de réception. Les requêtes sont bufferisées et le délai d'exécution d'une commande ne bloque que le joueur concerné.

`x` et `y` indiquent les dimensions du monde.

III.6 Le temps

Aucune attente active n'est toléré. Il ne doit pas y avoir de bloquage lorsque les clients sont stoppés, ou dans n'importe quelle phase du jeu

Les Trantoriens ont adoptés une unité de temps internationale. L'unité de temps est la seconde. Si t = 1, 'avance' prend 7 secondes. On choisira par défaut, t = 100. t est un entier.

Le référentiel de temps est le temps absolu.

III.7 Dénomination des objets

Seul la classe des objets est identifiable. Il est ainsi impossible de distinguer deux objets de la même classe. Par exemple, deux siburs auront la même dénomination puisqu'ils appartiennent à la même classe.

III.8 Reproduction des joueurs

Un joueur peut se reproduire grâce à la commande fork. L'exécution de cette commande entraîne la production d'un oeuf. Dès la ponte le joueur l'ayant pondu peut vaquer à ses occupations en attendant qu'il éclose. Lors de l'éclosion de l'oeuf, un nouveau joueur en sort, il est orienté au hasard. Cette opération autorise la connection d'un nouveau client. La commande connect_nbr renvoie le nombre de connections autorisées et non autorisées pour cette famille.

Délai de ponte : 42/t.

Délai entre la ponte et l'éclosion : 600/t.

III.9 Inventaire

Cette commandes permet de voir ce que le drone a comme objet et combien de temps il lui reste a vivre. Le seveur enverra par exemple la ligne suivante :

{nourriture 345, sibur 3, phiras 5, ..., deraumere 0}

III.10 Le broadcast

Pour émettre un message, le client doit envoyer au serveur la commande suivante :

```
broadcast <texte>
```

Le serveur, lui, enverra à tous les clients la ligne suivante :

```
message <K>,<texte>
```

avec K indiquant la case d'où provient le son.

III.11 L'Expulsion

Le drone à la faculté d'expulser tous les drones partageant la même unité de terrain. Il les pousse dans la direction qu'il regarde. Lorsqu'un client envoie au serveur la commande expulse, tous les clients partageant cette case recoivent la ligne suivante :

```
deplacement <K>\n
```

avec K indiquant la direction de la case d'où le drone provient.

III.12 Interface graphique

Le projet devra être muni d'un client de visualisation graphique. Celui-ci devra proposer une représentation en temps réel du monde tel qu'il est sur le serveur.

L'interface doit intégrer au minimum une visualisation 2D par l'intermédiaire d'icônes, permettant ainsi une représentation du monde. L'interface 3-D ou tout autre type de représentation sera un atout appréciable au projet. Il faut de plus inclure la visualisation des sons.

Il sera réalisé en C, en PHP, en Perl, en pyhton, etc... (tant que ça fonctionne sur les dumps), et communiquera en réseau avec le serveur pour récupérer le contenu de la map, des équipes, des inventaires, etc... bref tout ce qu'il faut pour pouvoir voir ce qu'il se passe dans le jeu.

Le but du client graphique est de pouvoir suivre des matchs, donc connaître la progression de chaque joueur, chaque équipe, qui est en avance, qui a gagné, etc... Vous devez donc concevoir son interface utilisateur dans cette optique.

Chapitre IV

Consignes

IV.1 rendu

Ce projet ne sera corrigé que par des humains. Vous êtes donc libres d'organiser et nommer vos fichiers comme vous le désirez, en respectant néanmoins les contraintes listées ici.

Le binaire du serveur doit se nommer `serveur`, celui du client `client`, et celui du client graphique `gfx`

Vous devez rendre un Makefile. Il devra compiler le serveur, le client, et le client graphique, et contenir les règles : serveur, client, gfx, all, clean, fclean, re. Il ne doit recompiler les binaires qu'en cas de nécessité.

Votre projet doit être à la Norme. Vous devez gérer les erreurs de façon raisonnée. En aucun cas votre programme ne doit quitter de façon inattendue (Segmentation fault, etc...).

Vous devez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant les logins de votre groupe suivis d'un '\n' :

```
$>cat -e auteur
xlogin$
ylogin$
zlogin$
$>
```

IV.2 Fonctions autorisées

Dans le cadre de votre partie obligatoire, vous avez le droit d'utiliser toutes les fonctions de la libc et tous les appels système.

ATTENTION: vous n'avez néammoins pas le droit d'utiliser de threads (dans le serveur) ni de sockets non bloquantes (donc pas de fcntl(s, O_NONBLOCK))

Vous avez l'autorisation d'utiliser d'autres fonctions dans le cadre de vos bonus, à condition que leur utilisation soit dûment justifiée lors de votre correction. Soyez malins.

Vous pouvez poser vos questions sur le forum dans l'intranet dans la section dédiée à ce projet.