MKAVAVO Eddin

THIRION Paul

Systèmes et réseaux

Rapport final

Table des matières

Introduction

Nous avons implémenté une version du jeu «6 qui prend ». Le but du jeu est de se débarrasser de toutes ses cartes en accumulant le moins de points (appelés ***tête de bœufs***) possibles. Dans ce rapport nous allons expliquer les étapes du déroulement d'une partie et le fonctionnement technique de notre projet.

Ce dernier est fait intégralement en Bash. Il ne fonctionne qu'en locale et permet de lancer des parties successives à 6 joueurs. La partie s'arrête lorsque l'un des 6 joueurs a atteint 66 points ou plus.

Organisation des données

Dans le cadre de notre projet, une partie est représenté avec un dossier et plusieurs fichiers des gestions.

Le fichier ***historique*** regroupe l'ensemble des plateaux de jeux qui existent au cours de la partie tour par tour.

Les fichiers portant le nom des joueurs sont en quelque sorte une fiche d'identité doublée d'un historique de joueur. Ils regroupent le nom du joueur, le PID de son terminal, les cartes qui lui ont été distribué, leur état, les cartes choisis à chaque phase de jeu, ainsi que l'évolution des points tour par tour.

Le fichier ***paquetCarte*** regroupe l'ensemble des cartes générés pour cette partie. Il est à noter qu'un nouveau paquet est généré lorsqu'une nouvelle manche se lance. L'ancien paquet est simplement renommé sous la forme ***paquetCarteMancheX*** avec X étant le numéro de l'ancienne manche.

Une carte est renseignée sous la forme l***$nb $etat $tete***. $nb est la valeur de la carte entre 1 et 104, ***$etat*** est sous la forme ***+*** ou ***-.*** Le premier étant l'état *disponible* le second l'état *indisponible*. Ainsi la carte ***l30 – 3*** décrit la carte 30 avec 3 têtes de bœufs dans l'état utilisé/indisponible.

Dans un fichier d'information l'information prendra généralement la forme ***manche$numM tour$numT $typeDeDonnee $donnee*** par *exemple* ***manche1 tour7 réponse 78*** indique que le joueur portant le nom du fichier a joué la carte 78 dans la manche 1 au tour 7.

Déroulement d'une partie classique

Lancement

Pour lancer le jeu il faut lancer le script ***leJeu.sh***. Ce script prend 3 paramètres. Le premier est le nombre de joueurs non-robot, le second est la valeur minimale d'une tête de bœuf, la troisième est la valeur maximale d'une tête de bœuf. S'il y a des joueurs non-robot ce script va demander les pseudos successivement. S'il n'y a que des robots les joueurs porteront des noms allant de J_bot1 à J_bot6. Par souci de gestion les pseudos commencent tous par ***J_.***

Initialisation des informations

leJeu.sh lance le scripts ***initPartie.sh***. Ce script génère le dossier de partie. Ce dernier est stocké dans le dossier ***LesParties*** et porte le nom ***P_nomJoueur1_ nomJoueur2_nomJoueur3_ nomJoueur4_nomJoueur5_ nomJoueur6*** le tout suivi par un numéro aléatoire compris entre 0 et 32767. Le cas ou une partie à exactement le même nom qu'une autre n'a pas été géré. Mais les probabilités que cela arrive est faible dans le cas d'un projet. Nous sommes conscients qu'il faudrait contrôler cela sinon.

initPartie.sh va lancer initCarte.sh en lui passant les têtes min et max qu'il a lui-même reçu de leJeu.sh une fois que initCarte.sh ce sera terminé, initPartie.sh affiche le nom du fichier de partie. Ce dernier sera récupéré par leJeu.sh

initCarte.sh a un script simple. Il va renseigner 104 cartes avec une valeur de tête entre min et max. Chaque ligne correspond à une valeur de carte. Ainsi la care à la ligne 23 aura la valeur 23. Seul la valeur de tête de bœuf est aléatoire. Pour éviter les confusions lors des filtres futurs, la valeur de carte est précédée d'un l.

Une fois toutes les informations généré, le jeu peu commencer. Cela se passe lorsque ***leJeu.sh*** lance ***gestionPartie.sh***. Ce dernier script va tirer les cartes du plateau et les écrire dans ***historique,*** il va aussi distribuer 10 cartes à chaque joueur. Les scripts ***tirCarte.sh*** et ***distribueCarte.sh*** seront utilisé. Le premier va récupérer une carte disponible dans le fichier paquetCarte, la rendre indisponible et l'afficher. L'affichage est récupéré pour mettre à jour historique. La seconde va faire la même chose mais 10 fois et mettre à jour non pas l'historique mais le fichier du joueur passé en premier paramètre

Une fois tout ceci fait gestionPartie.sh lance les joueurs dans de nouveaux terminaux avec la commande ***xterm -e***. Un terminal lance un ***joueur.sh*** en tâche de fond. joueur.sh est le script avec lequel le joueur interagit directement. Il est chargé de récupérer les entrées clavier d'un joueur et de suivre les ordres de gestionPartie.sh. La première chose que ce script fait est d'initialiser sont fichier dédié. A partir de là il mènera des actions de choix et de placement, ces actions sont détaillées plus tard dans le rapport.

Echanges

Il est primordial que gestionPartie.sh et les instances de joueurs communiquent entre eux. Cette communication s'effectue selon une dynamique de signaux et d'attente de signaux. gestionPartie possède un unique ***trap*** qui incrémente une variable ***$count***. Selon la valeur de cette variable et le moment, des actions seront prise. Par exemple après avoir créé tous les joueur.sh, le script attendra de capter 6 signaux réponses puis passera à la suite. gestionPartie emmétra un signal USR1 au changement de tour et de manche. Il emmétra un signal USR2 pour signal à un joueur que c'est à son tour de jouer pour poser sa carte. Après l'émission de ce signal USR2, gestionPartie.sh passe en attente.

De son côté joueur.sh possède 2 traps. Le premier capture le signal USR1 et sert à définir si le joueur est en attente ou non. Le second est lié au signal USR2 et sert à informer le script qui peut et doit poser la dernière carte qu'il a sélectionnée. Fut un temps il y avait des erreurs de signaux perdu. Nous supposons que cela était lié a des conflits d'écritures/lecture dans des fichiers plutôt qu'à la gestion de signaux car le problème semble ne plus être d'actualité depuis que d'autres ont été réglé.

joueur.sh emmétra un signal USR1 vers gestionPartie.sh lorsque sa carte sera choisie ou lorsqu'il aura placé sa carte après réception du signal USR2.

Evènements

Choix des cartes

Si le joueur est un humain, son terminal lui demandera de rentrer une valeur appropriée et l'enregistrera comme réponse. Une mauvaise valeur chiffrée sera refusée. Le script ne poursuivra que si une valeur correct est rentrée.

S'il s'agit d'un robot alors le choix se fera par l'intermédiaire du script ***choisi.sh***. Ce dernier attend 4 paramètres. Le nom du joueur, le type de choix à faire (0 = choix de cartes, autre = choix de ligne à récupérer), la manche, le tour.

Dans le cas d'un choix de carte, le script cherchera la carte jouable avec l'écart positif le plus petit avec l'une des dernières cartes du plateau de jeu. Si aucune carte n'est jouable alors il retournera la première carte avec la plus haute valeur de tête de bœuf qu'il possède. Par défaut le script choisit la première carte qu'il possède.

Dans le cas d'un choix de ligne, le script cherchera la ligne avec la valeur cumulée de tête de bœuf la plus petite. Par défaut le script choisit la ligne 1.

Placement des cartes

Pour que le joueur face le choix dans son terminal, il a fallu que chaque joueur place sa propre carte. Ainsi c'est gestionPartie.sh qui envoie un signal aux joueurs dans l'ordre croissant des valeurs des cartes sélectionné. Ce signal va faire exécuter ***placeCarte.sh*** à chaque joueur.

placeCarte.sh attend 4 paramètres. Le nom du joueur, le numéro de la carte, la manche et le tour. Dans un premier temps le script va essayer de trouver un endroit ou placer la carte selon la configuration actuel du terrain. Trois événements peuvent se déclencher.

1. Un emplacement a été trouvé mais il y a déjà 5 cartes sur cette ligne. Dans ce cas le script va appeler recupCarte.sh pour faire récupérer chaque carte au joueur (*recupCarte.sh est un petit script qui met à jour les informations du joueur selon les informations de la carte dont le numéro est passé en paramètre*). Une fois cela fait il remplace la ligne par la carte du joueur.

2. Un emplacement a été trouvé et il n'est pas plein. Dans ce cas la carte est simplement posée à la suite et l'historique mis à jour.

3. La carte ne peut pas être posée. Dans ce cas il est demandé au joueur quelle ligne il veut récupérer. Si le joueur est un robot, il appellera le script choisi.sh. De la même façon que pour le choix de la carte, un garde-fou est placé pour prévenir les mauvaises entrées chiffrées.

Ces étapes de choix et de gestion vont se répéter jusqu'à ce que l'un des joueurs ait atteint ou dépassé 66 points. A noter que quoi qu'il arrive les 10 tours d'une manche se joueront. Ainsi il est possible de rattraper son retard tant que la manche n'est pas terminée.

Fin de partie

Lorsque la condition de boucle est atteinte, gestionPartie.sh récupère les points finaux de tous les joueurs et créer un fichier de classement trié. La ligne ***END*** est ajoutée à historique. Cette ligne est nécessaire car dans sa période d'attente, chaque joueur vérifie si l'historique contient cette ligne. S'il la contient alors cela marque leur condition de fin et les fait sortir. Par la suite si le joueur n'est pas un robot, les résultats lui sont affiché.

Le jeu se termine alors et une nouvelle partie peut être lancé.

Discussion

Ce qui a été ajouté depuis la présentation

Au moment de la présentation les robot faisaient des choix aléatoires. Ceci a été corrigé grâce au script choisi.sh. Les robots chercheront à jouer de façon plus stratégique et chercheront à faire perdre les autres en plaçant des cartes avec une haute valeur de tête de bœuf.

Le classement de fin ne s'affichait que dans la fenêtre principale. Cela a été corrigé.

Le script gestionJoueur.sh a été renommé joueur.sh pour éviter les confusions.

Ce qui n'a pas pu être fait

L'ajout d'un peu de code en C. Le classement aurait pu être réécrit en C pour la forme, mais cela semblait contreproductif de détruire un code qui existe et fonctionne correctement dans l'unique but de diversifier les langages.

Ce que nous aurions voulu ajouter

Un grands regret est que le projet ne fonctionne qu'en local. Une version en réseau était à notre porté en terme technique, mais le temps nous a fait défaut. Nous aurions pu utiliser ***nc*** pour mettre cela en place, ou partir sur une architecture plus hybride avec des sockets en C.

Nous aurions voulu mettre en place un system de compte d'utilisateur pour stocker les informations des joueurs et conserver une certaine continuité dans les jeux. Avec des statistiques personnels, un peu à l'image du site liChess.

Une dernière chose aurait été d'avoir plusieurs types de robot avec des stratégies différentes et des changements en cours de jeu. Par exemple avec un bot agressif, un autre défensif et un dernier sournois.

Conclusion

Le projet était très intéressant et nous avons apprécié le faire. Particulièrement en partant sur une architecture purement bash. Nous avons appris de nombreuses choses et nous rendons compte qu'il en reste encore énormément. Nous regrettons tout de même d'avoir un peu délaissé la partie réseau de ce projet.