Système et réseaux : rapport

Valentin VERSTRACTE & Evan PETIT

L3 — December 4, 2021

Table des matières

Rappel des règles

Le jeu se joue en plusieurs manches. Lors de la première manche, chaque joueur reçoit une carte dont il prend connaissance. Ces cartes représentent les chiffres de 1 à 100. Le jeu est coopératif, il suffit de jouer les cartes par ordre croissant. De la plus petite à la plus grosse. C'est vraiment idiot. Mais voilà, le truc c'est que personne ne doit se parler, ni se faire des signes. Il n'y a donc pas d'ordre de jeu. Celle ou celui qui pense avoir la plus petite carte la pose. La ou le suivant qui pense avoir la plus petite suivante la pose et ainsi de suite

Introduction

L'application est découpé en 3 programmes shell. GestionJeu, JoueurHumain, JoueurRobot. Chacun d'eux utilisent des terminaux pour afficher l'état du jeu ou demander des entrées. Nous allons tout d'abord présenter comment les scripts communiquent et ensuite la structure.

1 La communication

Il existe dans ce projet plusieurs types de communications. On utilise les pipes pour communiquer des données et notifier des actions et les fichiers tmp pour synchroniser des données entre les shell et subshell

1.1 Les pipes : une micro API

Chaque terminal utilisent constamment des pipes pour recevoir ou envoyer des données / actions. Il a été choisit de construire une micro API pour facilité cette communication. On envoie un message dans les pipes constituer de la forme suivante :

[n°action];[message]

Le ";" permet de facilité la séparation et récupération du n°action et du message. Voici les n°action que l'on peut trouver:

- N°1 décrit que la carte trouvée était la bonne.
  Le message sera du type : *Bravo, une carte a été trouvée, voici les cartes trouvées : (1 2)*
- N°2 décrit que la carte trouvée était mauvaise.
  Le message sera du type : *Perdu, la carte 2 n'était pas la bonne, la bonne était : 1. On recommence !*
- N°3 décrit que le tour est terminé, on passe au suivant.
  Le message sera du type : *Félicitations, le tour n°'2 est terminé, on passe au tour suivant*
- N°4 décrit que le jeu est terminé, on passe au suivant.
  Le message sera du type : *Félicitations, le jeu est terminé*
- N°5 décrit que le joueur a reçu toutes ses cartes dans son fichier tmp
- N°9 est spécifique au robot. Il répond à un problème technique. Brièvement à chaque fois qu'une carte est jouer le robot s'envoie à lui même le message "9;distance" au bout d'un certains temps. "distance" étant la distance entre sa carte la plus faible et la dernière carte jouée. Comme ça si une nouvelle carte est jouée entre temps, la distance change et on laisse au joueur le temps de réévaluer la situation.

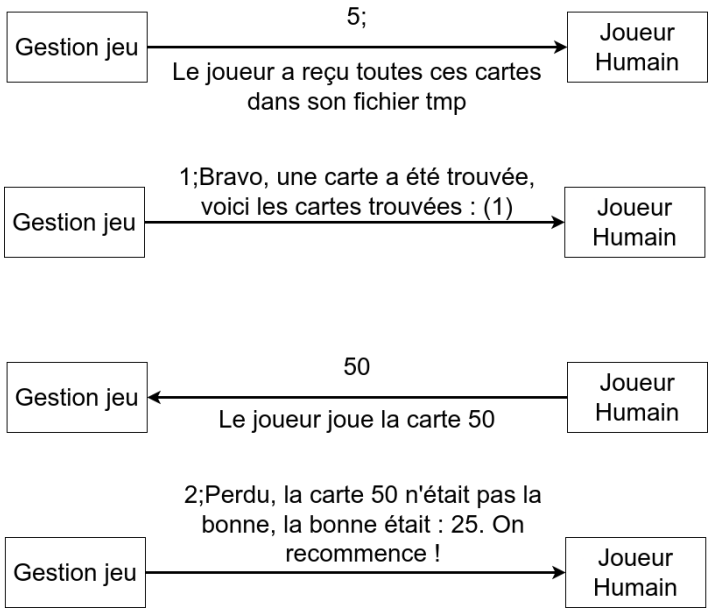

Figure 1: Exemple d'une communication avec la micro API

On remarquera qu'il suffit d'envoyer un chiffre à gestion jeu pour qu'il soit interpréter comme une carte jouer. Si l'application était beaucoup plus important, cela pourrait représenter une faille de sécurité importante.

1.2 Les fichiers tmp

Les fichiers tmp sont présent pour résoudre le problème de synchronisation des variables entre le shell et le subshell (problème dû aux commandes bloquantes). Chaque ligne du fichier tmp représente une carte. On lit toutes les lignes pour savoir les cartes du joueur. Ainsi, pour transmettre des cartes, gestion jeu écrit dans le fichier tmp et envoit un message à travers les pipes pour décrire quand il a finit. Quand le joueur joue une carte, on réécrit toutes les cartes ligne à ligne dans le fichier sans celle jouée.

2 Structure générale de l'application

2.1 Gestion Jeu

2.1.1 L'initialisation

Une première phase de l'initialisation définit le nombre de joueur. Elle le fait en demandant le nombre de joueur humain puis le nombre de joueur robot. On lance en conséquence les scripts nécessaire au travers d'un terminal xTerm. Chaque joueur humain obtient un terminal au travers du quel il peut interagir.

Une deuxième phase de l'initialisation s'occupe de chercher le nombre maximum de tour / carte maximum que l'on peut distribuer. Il s'agit d'un petit algorithme qui incrémente une variable en fonction d'une condition. Cette condition est la suivante : si la variable multiplier par le nombre de joueur est inférieur à 100, alors on incrémente. On continue jusqu'à que la multiplication soit supérieur à 100. On en déduit à la fin le nombre maximum de tour.

La troisième phase de l'initialisation appel la fonction qui gère les cartes. Nous allons décrire cette fonction dans la sous section suivante puisque elle est appelé à différent moment de la partie.

2.1.2 La gestion des cartes

La gestion des cartes a trois objectifs :

- Mélanger aléatoirement les cartes
- Distribuer aux joueurs les cartes de 0 à N (N = indice du rond multiplier par le nombre de joueur)
- Trier les cartes distribuer pour savoir dans quel ordre elles doivent être jouer

Ces trois objectifs devant être atteint de manière récurrente dans le code, ils sont contenu dans une fonction. Cette fonction est appelé après chaque ronde, à chaque fois qu'une mauvaise carte est joué et lors de l'initialisation.

Le troisième objectif est motivé par une idée simple : on souhaite juste savoir la carte qui doit être jouer sans savoir quel joueur doit la jouer.

2.1.3 La gestion du jeu

À partir de l'API nous pouvons savoir une chose importante : une carte a été jouer et on connaît son nombre. C'est ici que tout prend son importance car nous allons effectuer en conséquences des traitements à partir de cette action.
On regarde tout d'abord si la carte jouer est la carte qui devait être jouer (merci l'objectif 3 de la gestion de cartes). Si ce n'est pas le cas, on notifie les joueurs que la carte tirée est mauvaise et on redistribue des cartes (fonction gestion des cartes) et le tour recommence. Sinon, on notifie tout les joueurs de la carte trouver. On regarde ensuite si il s'agissait de la dernière carte du tour. Si c'est le cas et qu'il reste un tour, on passe au tour suivant en redistribuant des cartes.

2.2 Joueur Humain

Le comportement du script de joueur humain est scindée en deux grandes parties. La première s'occupe de la réception des communications avec gestion jeu et de l'affichage. La deuxième de l'entrée utilisateur et de l'envoie de la carte jouée. Cette structure de code n'est pas venu naturellement. Elle répond à problème technique : chaque partie utilise une commande bloquante.

2.2.1 Le problème des commandes bloquantes

On se retrouve avec deux commandes bloquantes. Une première essaye de lire les pipes, une deuxième essaye de lire l'entrée utilisateur. Pour résoudre ce problème nous avons choisit la solution suivante : l'entrée utilisateur sera lu par un subshell et les pipes seront lui à partir du shell. Un deuxième problème technique est alors apparût. Les variables entre le shell et le subshell ne sont pas synchronisées. Ceci représente une difficulté puisque nous avons besoin de savoir les cartes reçus depuis les pipes dans l'entrée utilisateur (pour savoir si l'utilisateur peut bien joué cette carte). Cette difficulté a été résolu a l'aide des fichiers TMP.

2.2.2 Réception des communications(API) / affichage

La partie centrée sur la réception des communications et l'affichage lit constamment les pipes. En fonction du message reçu, une action en découle. Il peut très bien s'agir de lancer le subshell pour lire les entrées utilisateur ou afficher les informations reçues

2.2.3 L'entrée utilisateur

La partie centrée sur l'entrée utilisateur n'est pas très compliquer. Elle lit tout le temps le terminal. On est alors obliger de trier l'affichage et l'entrée utilisateur. Comme on souhaite lire la carte que l'utilisateur souhaite jouer et que l'affichage n'est jamais juste un chiffre on regarde si la lecture est un chiffre. Si c'est le cas on récupère les cartes de l'utilisateur dans son fichier tmp associé et on regarde si la carte voulant être jouer existe. Si c'est le cas, on la retire du fichier et on envoie la carte jouer à gestion jeu.

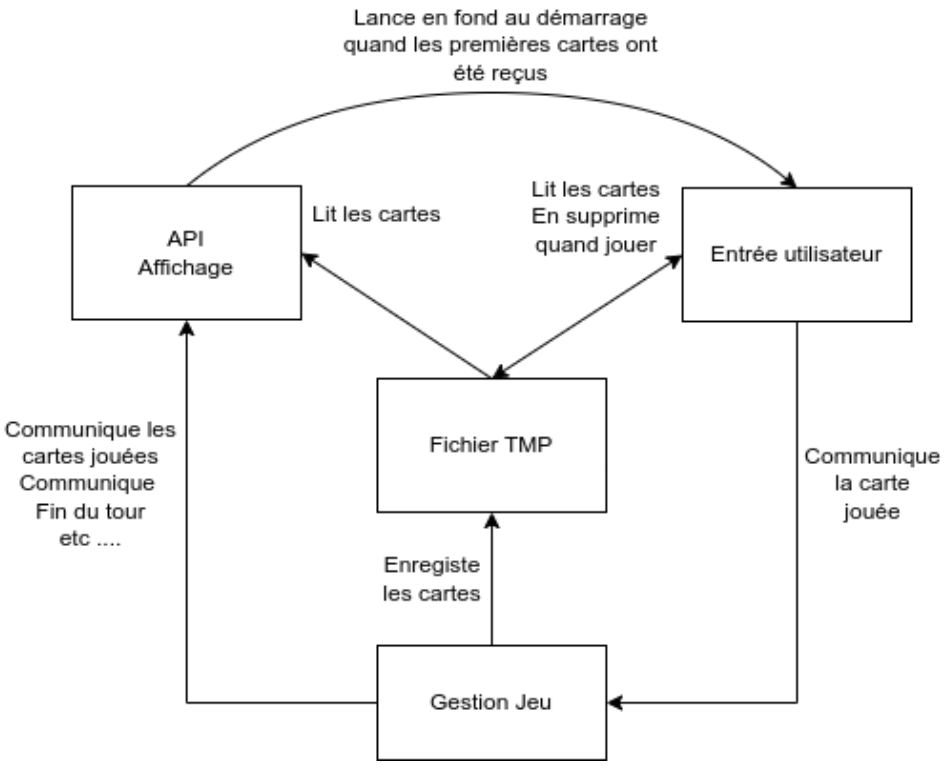

Figure 2: Illustration / cycle de vie de joueur humain

2.3 Joueur robot

Le premier réflexe a été de copier coller le code du joueur humain et de remplacer les entrées utilisateurs par une stratégie robot.

2.3.1 La stratégie

Il a été difficile d'établir une stratégie pour le robot. Le problème a donc été décalée et on se repose sur le joueur. L'idée est simple. Quand le joueur estime que ce n'est pas son tour, il va laisser jouer. Ainsi, au bout d'un certains temps, le robot joue sa carte la plus faible. Malgré des faiblesses cette stratégie reste très forte car repose sur l'intelligence du joueur humain.

2.3.2 Réception des communications(API) / affichage

rfrfrf

2.3.3 L'astuce de la distance

fefef