Jeu de Morpion en réseau - Documentation

Introduction

Ce jeu de morpion est un jeu de plateau pour deux joueurs dans lequel chaque joueur place un symbole (X ou O) sur une grille. Le but est d'aligner cinq symboles identiques horizontalement, verticalement ou en diagonale. Si la grille est complétée sans qu'il y ait de vainqueur, il y a match nul.
Dans cette version jouable dans le terminal en réseau développée en **python**, les joueurs peuvent jouer en même temps même s'ils ne sont pas sur la même machine. Ils doivent se connecter à un serveur s'occupant de la gestion du jeu pour jouer

Développement

Partie réseau

Fonctionnement global

Ce jeu utilise le schéma client-serveur pour établir la communication des informations entre les joueurs et l'évolution de la partie. Il utilise le protocole UDP pour le transport de messages.

Côté serveur :

- Gère l'enregistrement des joueurs
- Gère la mécanique de jeu (cf. partie gameplay)
- Gère l'envoi et la réception des messages pour récupérer les coups à jouer
- Enregistre les joueurs dans un dictionnaire d'adresse de la forme : `{<addr>:(<pseudo>, <valeur du symbole (1 ou 2)>, <symbole X ou O>)}`

Côté client

- Gère l'arrivée d'un joueur et fait la requête de son enregistrement auprès du serveur
- Un thread dédié à la réception (un receiver) affichant les données reçues du serveur
- Un thread dédié à l'envoi envoyant au serveur les données entrées dans le terminal par l'utilisateur

Connexion

La connexion au serveur par un client passe par différentes étapes gérées par la fonction `player_arrival(data, addr)` avec `data` le pseudo du nouvel utilisateur et `addr` son adresse côté serveur, et par la fonction `new_user_handler()` côté client.

Ce tableau décrit les étapes de la connexion d'un client :

Étape	Description	Exécution
Demander un pseudo à la connexion	Demander un pseudo au client lors de sa connexion et l'envoyer au serveur	Client
Vérifier que que la partie n'est pas déjà pleine	Vérifier que le partie n'est pas déjà pleine (2 joueurs sur 2) en analysant la taille du dictionnaire contenant les adresses des joueurs. Si la partie est pleine, envoyer le message `game_already_full` au client et refuser sa connexion (ne pas l'ajouter dans le dico des adresses)	Serveur
Répondre quant à la validité du pseudo	Vérifier que le pseudo n'est pas déjà pris par un joueur déjà connecté. - Si le pseudo est déjà dans le dico des adresses, envoyer le message `invalid_name` au client, l'informant que son pseudo n'est pas valide - Si le pseudo n'est pas dans le dico, envoyer **le pseudo validé** au client en guise de réponse positive, l'informant que son pseudo est valide	Serveur
Analyser la réponse reçue du serveur	- Si le client reçoit `game_already_full`, le prévenir que la partie est déjà pleine et stopper la communication - Si le client reçoit `invalid_name`, redemander un pseudo différent à l'utilisateur - Sinon, le serveur a validé le pseudo et la connexion. Sortir de la boucle demandant un pseudo.	Client

Échanges serveur/ client

Une fois un client enregistré auprès du serveur (dont l'adresses est dans le dico), celui ci est libre d'envoyer des messages à tout moment. Tous ces messages ne doivent pas être considérés comme des coups à prendre en compte dans la partie de morpion à chaque fois.
Ainsi, à chaque message reçu, le serveur doit analyser le contexte du message et sa validité afin de décider si oui ou non, le message reçu doit être considéré comme un coup valable.
Pour cela, on implémente une fonction `player_move_handler(move, addr)` prenant en paramètres le coup `move` joué par le joueur d'adresse `addr`.

On observe ainsi cette série de conditions au sein de la fonction :

- ▼ La partie a bien commencé
  - ▼ C'est bien son tour de jouer
    - ▼ Il entre un coup existant (mais pas forcément jouable, **la suite sera analysée par la partie gameplay**)
      - ▼ Le coup est jouable

        Jouer le coup dans la partie et envoyer à tous les clients la grille actualisée et le nouveau joueur courant
      - ▼ Le coup n'est pas jouable

        Prévenir que le coup n'est pas jouable (car il y a déjà un pion dessus) et redemander de jouer
    - ▼ Il entre un coup qui n'existe pas ou un envoie un message quelconque

      Prévenir le joueur que ce coup n'est pas valable par le message "Coup invalide !" et lui redemander de jouer
  - ▼ Ce n'est pas son tour de jouer

Le prévenir que ce n'est pas son tour de jouer (et ne pas impacter la partie)

▼ La partie n'a pas commencé

Nous sommes dans le cas où les deux joueurs ne sont pas connectés (seul un des deux est connecté, et celui ci essaye de jouer).
Prévenir le joueur qui est déjà connecté que le partie n'a pas encore commencé (en se basant sur la longueur du dico des adresses)

Déconnexion

On considère 3 cas de fin du jeu (sachant que la seule fin attendue dans le cahier des charges est celle liée à la victoire d'un joueur ou à un match nul)
Pour cela, on implémente une fonction `end_game(reason, notable_addr)` prenant en argument la raison de la fin de la partie ainsi que l'adresse du joueur ayant causé la fin de la partie (soit par sa victoire, soit par son départ) ou rien si la raison est autre (mais ce n'est pas censé arriver dans notre cas)

Ce tableau décrit les 3 types de fin de partie possibles :

Type de déconnexion	Reason	Description	Gestion
Fin de partie	`game_done`	Victoire de l'un des joueurs	- Prévenir tous les joueurs du résultat de la partie et stopper la partie
Match nul (draw)	`draw`	Match nul (la grille est pleine et personne n'a gagné)	- Prévenir tous les joueurs du résultat de la partie et stopper la partie
Client	`player_left`	Un client envoie "quit".	- Prévenir tout le monde de la déconnexion de l'un des joueurs - Prévenir le joueur ayant quitté qu'il a perdu "par forfait" - Prévenir les autres de la victoire "par forfait" du joueur restant
Serveur	`server_stopped`	Le serveur ferme (pas censé arriver dans notre cas).	- Prévenir tout le monde de la fermeture (liée à une erreur) du serveur

Partie gameplay

Du côté du gameplay, la partie est hébergée sur le serveur et se joue dessus. Les client ne font que recevoir l'état de la partie à chaque actualisation.

Lorsqu'un coup est considéré comme valable par le serveur (cf. partie réseau, échanges serveur/client), celui-ci est analysé au sein de la fonction `player_move_handler` décrite plus haut en suivant ces étapes :

▼ Si le coup est jouable (il n'y a pas déjà de pion ici) : vérifié avec la fonction `can_play(board, move)`

▼ Jouer le coup : avec la fonction `play_move(board, move, player)`

Envoyer à tous la grille dans son nouvel état

▼ Si ce coup entraîne la victoire du joueur courant : vérifié avec la fonction `check_win(board, player)`

Lancer la fonction `end_game(reason, notable_addr)` avec en argument `("game_done", <joueur_courant>)`

▼ Si ce coup entraîne un match nul (mais pas une victoire) : vérifié avec la fonction `check_draw(board)`

Lancer la fonction `end_game(reason, notable_addr)` avec en argument `("game_done", None)`

▼ Si le coup n'est ni un match nul ni une victoire

Passer au joueur suivant en changeant le joueur courant et lui demander son prochain coup

▼ Si le coup n'est pas jouable

Prévenir le joueur que son coup est invalide et le laisser rejouer jusqu'à ce qu'il soit valable

Auteur

- **Balthazar GIROT**, CUPGE 1 ESIR (2022-2023), dans le cadre d'un projet individuel consistant à implémenter en python un jeu de morpion jouable dans le terminal en réseau