UBFC
UNIVERSITÉ
BOURGOGNE FRANCHE-COMTÉ

Rapport du projet :

Système et Réseaux

Wassim Ennaji
Choukri Imane

Sommaire

Introduction

Dans ce projet, notre objectif est d'illustrer l'utilisation des concepts et mécanismes abordés lors du cours de Systèmes & Réseaux en développant une version informatique du jeu de cartes "6 qui prend."
On cherchera ainsi à permettre aux utilisateurs de s'engager dans des parties via un terminal, interagissant avec d'autres joueurs et/ou des processus robots.
Le "6 qui prend" est un jeu de cartes de type "défausse" créé par Wolfgang Kramer en 1994, où il faut tactiquement placer ses cartes dans différentes rangées sans jamais poser la sixième. Chaque manche se compose de 10 tours, avec pour objectif ultime d'avoir le moins de "têtes de bœuf" à la fin de la partie.

Le gestionnaire du jeu assure l'accueil des joueurs, la distribution des cartes, la gestion des rangées, le comptage des têtes de bœuf par joueur, etc.
Les joueurs humains interagissent avec le jeu via le terminal, tandis que les joueurs robots tentent de minimiser leurs têtes de bœuf. Le processus de génération de statistiques échange des informations avec le gestionnaire pour produire des statistiques, des points, et même sauvegarder les parties dans des fichiers de log, générant un document PDF illustrant le déroulement de la partie.

Notre mise en œuvre du jeu permettra à des joueurs humains et robots de participer sur la même machine, respectant ainsi scrupuleusement les règles du "6 qui prend." Nous explorerons également des fonctionnalités avancées, telles que le jeu en réseau, des joueurs robots plus intelligents, et le jeu en équipes, pour enrichir davantage l'expérience ludique. Ce rapport documentera le processus de conception, nos choix techniques, et les résultats obtenus tout au long du développement de cette simulation informatique du "6 qui prend." Nous aspirons ainsi à fournir une expérience interactive et divertissante tout en mettant en pratique les principes fondamentaux des systèmes et réseaux.

Le gestionnaire du jeu

Son rôle est d'initialiser et de superviser une partie, gérant la connexion des joueurs, la distribution des cartes, le suivi des scores, et utilisant un script Bash pour mettre à jour les résultats dans un fichier texte et générer un rapport PDF.
Chaque joueur est géré par un thread distinct avec des mutex pour la synchronisation. Le jeu se déroule en plusieurs tours jusqu'à l'atteinte d'un score cible ou la fin du nombre prédéfini de manches.

Structures :

Carte : Représente une carte à jouer avec une valeur numérique, une valeur en têtes de bœuf, et l'identifiant du joueur à qui elle est attribuée.
Joueur : Contient les cartes en main d'un joueur et son score.
Rangee : Représente une rangée de cartes dans le jeu avec le nombre total de têtes de bœuf des cartes y présentes.
Partie : Représente l'état complet du jeu, incluant les joueurs, les équipes, les rangées et l'état actuel de la partie.
ThreadArgs : Contient des informations pour les threads des joueurs, y compris le socket, la partie, et le nombre de joueurs.

Fonctions :

initialiserPartie : Initialise les scores des joueurs et l'état initial de la partie.
initialiserCarte : Crée un paquet de cartes pour le jeu.
initialiserCartesDistribuees : Initialise un tableau pour suivre les cartes distribuées.
melangerCartes : Mélange le paquet de cartes.
distribuerCartes : Distribue un ensemble de cartes à un joueur.
viderrangee : Vide une rangée de toutes ses cartes.
intitialiserRanger : Initialise les rangées avec une carte au début de la partie.
affichertable : Affiche l'état actuel des rangées de cartes.
recupererpaquet : Récupère les cartes non distribuées du paquet.
serializerpaquet : Convertit une carte en un format de données pour le transfert sur le réseau.
envoyerPaquetAuJoueur : Envoie un paquet de cartes à un joueur via le réseau.

deserializecarte : Convertit les données reçues du réseau en une structure de carte.
recevoirCarte : Reçoit une carte envoyée par un joueur.
obtenirIndiceRaneeAvecTeteDeBoeufMinimum : Trouve la rangée avec le nombre minimum de têtes de bœuf.
obtenirTeteDeBoeufMinimum : Calcule le nombre minimum de têtes de bœuf parmi toutes les rangées.
inferieur : Vérifie si la carte d'un joueur est inférieure à toutes les cartes dans les rangées.
peutEtrePlaceeDansRangee : Détermine si une carte peut être placée dans une rangée donnée.
peutEtrePlaceeDansUneRangee : Vérifie si une carte peut être placée dans l'une des rangées disponibles.
obtenirIndiceRangeePlacable : Trouve une rangée où une carte peut être placée.
serializetable : Convertit l'état d'une rangée en format de données pour le réseau.
envoyerTableAuxJoueurs : Envoie l'état actuel des rangées à tous les joueurs.
estRangeePleine : Vérifie si une rangée est pleine.
placerDansRangee : Place une carte dans la rangée appropriée.
pleinrangee : Vérifie si la rangée cible pour une carte est pleine.
envoyerScore : Calcule et envoie le score aux joueurs après chaque tour.
envoyerMessageATousLesJoueurs : Envoie un message à tous les joueurs.
envoyerMessageAuJoueur : Envoie un message à un joueur spécifique.
trierCartesCroissant : Trie les cartes en ordre croissant.
determinerJoueurGagnant : Détermine le joueur avec le score le plus bas (gagnant).
joueurAtteintScore : Vérifie si un joueur a atteint un score cible.
finPartieAuJoueur : Envoie un signal de fin de partie aux joueurs.
envoyernbrJoueurs : Envoie le nombre de joueurs à tous les joueurs connectés.
envoyernbrManche : Envoie le nombre de manches à tous les joueurs connectés.
envoyerscoregagnant : Envoie le score du joueur gagnant à tous les joueurs.

classementJoueurCroissant : Classe les joueurs en fonction de leur score en ordre croissant.
threadJoueur : Fonction exécutée par chaque thread de joueur pour gérer la logique de jeu spécifique à chaque joueur.
main : Fonction principale qui configure le gestionnaire, accepte les connexions des joueurs et lance les threads de jeu.

Fonctionnement:

Ce programme est un serveur fonctionnant sur un modèle client-serveur.

- **Initialisation du Serveur :** Le programme serveur démarre et attend qu'un nombre requis de joueurs se connecte à lui.
- **Initialisation de la Partie :** Une fois le nombre requis de joueurs connectés, le gestionnaire initialise la partie en créant les joueurs, les équipes et les rangées de cartes.
- **Déroulement du Jeu :** Le gestionnaire gère le déroulement du jeu en plusieurs tours. Il distribue les cartes, reçoit les choix des joueurs et détermine le placement des cartes sur le plateau de jeu.
- **Calcul des Scores :** Après chaque tour, le gestionnaire calcule et met à jour les scores des joueurs. Les scores sont enregistrés dans un fichier texte à l'aide d'un script Bash.
- **Script de Mise à Jour des Scores :** Un script Bash est utilisé pour mettre à jour les scores. Il vérifie l'existence du fichier de scores et le crée s'il n'existe pas. Le script nécessite certains arguments, tels que le numéro de la manche et les scores des joueurs.
- **Génération de Rapport PDF :** Les scores de chaque joueur sont ajoutés au fichier de scores, puis le script utilise pdflatex pour générer un document PDF reprenant les statistiques de la partie, y compris le contenu du fichier de scores. Ce PDF est déplacé vers un répertoire spécifié.
- **Gestion des Joueurs :** Chaque joueur est géré par un thread séparé sur le gestionnaire. Des mutex sont utilisés pour synchroniser l'accès aux données partagées.
- **Fin du Jeu :** Le jeu continue jusqu'à ce qu'un score cible soit atteint ou jusqu'à la fin du nombre prédéfini de manches. Le joueur avec le score le plus bas est déclaré gagnant, et un classement final est affiché.
- **Nettoyage et Fermeture :** À la fin du jeu, le gestionnaire effectue un nettoyage, ferme les connexions et libère toutes les ressources utilisées.

Les joueurs humains

Après s'être connecté au gestionnaire, le joueur entre dans la phase de jeu où il reçoit un paquet de cartes et l'état actuel des rangées de cartes du plateau. Les interactions du joueur, telles que la sélection et l'envoi de cartes au gestionnaire, sont gérées par le joueur. Il reste constamment informé des actions des autres joueurs et des changements sur le plateau. En parallèle, le joueur met à jour le score du joueur tout au long de la partie. À la fin de chaque manche, il vérifie les conditions de fin de partie, affichant des messages appropriés en cas de victoire ou de défaite. Une fois la partie terminée, le joueur se déconnecte du gestionnaire, marquant la fin de son exécution.

Structures :

Carte : Représente une carte à jouer avec une valeur numérique, une valeur en têtes de bœuf, et l'identifiant du joueur possédant la carte.
Rangee : Représente une rangée de cartes dans le jeu, avec le nombre total de cartes et de têtes de bœuf.
Partie : Contient l'état actuel du jeu, y compris les rangées de cartes et l'état de progression.
Score : Stocke le score d'un joueur.

Fonctions :

communiquerAvecServeur : Gère la communication avec le serveur, recevant des messages et les affichant.
choisirEtSupprimerCarte : Permet au joueur de choisir une carte à jouer et la supprime de son paquet.
affichepaquet : Affiche les cartes restantes du joueur.
envoyerCarte : Envoie une carte choisie au serveur.
initialiserScore : Initialise le score du joueur à zéro.
intialiserPartie : Initialise l'état de la partie.
afficherMessageErreur : Affiche des messages d'erreur et termine le programme.
deserializepaquet : Convertit les données reçues du réseau en une structure de carte.

libererPaquet : Efface les informations des cartes du paquet.
affichecarte : Affiche une carte individuelle.
recevoirPaquet : Reçoit un paquet de cartes du gestionnaire.
deserialiserScore et **recevoirscore** : Convertit les données reçues du réseau en un score et l'ajoute au score actuel du joueur.
recevoirCartesDesAutresJoueurs : Reçoit les cartes jouées par les autres joueurs.
deserializetable et **recevoirTable** : Convertit les données reçues du réseau en l'état des rangées de cartes et met à jour l'état de la partie.
affichertable : Affiche l'état actuel des rangées de cartes.
serializercarte : Convertit une carte en un format de données pour le transfert sur le réseau.
recevoirPartieTermineeDuServeur : Vérifie si la partie est terminée.
recevoirNbrJoueurs et **recevoirNbrManche** : Reçoit le nombre de joueurs et le nombre de manches du serveur.
recevoirScoreGagnant : Reçoit le score du joueur gagnant du gestionnaire.
recevoirMessageDuServeur : Reçoit un message textuel du serveur.
jeu_thread : Logique principale du jeu gérée dans un thread. Connecte le joueur au gestionnaire, gère les interactions du jeu, et met à jour l'état de la partie.
main : Point d'entrée du programme joueur. Initialise la connexion avec le gestionnaire et crée un thread pour gérer le jeu.

Fonctionnement:

Le programme joueur est spécialement conçu pour participer à un jeu de cartes en réseau:

- **Connexion au gestionnaire :** Lors du démarrage, le joueur établit une connexion avec le gestionnaire en utilisant une adresse IP et un port spécifié.
- **Entrée dans la Phase de Jeu :** Une fois la connexion établie, le joueur entre dans la phase de jeu, qui est gérée par un thread distinct pour permettre une gestion parallèle des opérations.
- **Réception des Données Initiales :** Au début de la partie, le joueur reçoit un paquet de cartes et l'état actuel des rangées de cartes sur le plateau de jeu depuis le gestionnaire.

- **Sélection et Envoi de Cartes :** Pendant la partie, le joueur choisit une carte à jouer. Cette sélection est ensuite envoyée au gestionnaire pour tenir à jour l'état du jeu.
- **Réception des Mises à Jour :** Le joueur reste en constante réception des mises à jour, notamment les cartes jouées par d'autres participants et les changements sur le plateau de jeu.
- **Mise à Jour du Score :** Le score du joueur est mis à jour en continu tout au long de la partie, reflétant les actions et performances du joueur.
- **Vérification des Conditions de Fin de Manche :** À la fin de chaque manche, le joueur évalue les conditions de fin de partie, basées sur le score ou le nombre de manches jouées.
- **Affichage des Résultats :** Des messages indiquant la victoire ou la défaite sont affichés selon l'issue de la partie, fournissant une rétroaction au joueur.
- **Déconnexion du gestionnaire :** Une fois le jeu terminé, le joueur se déconnecte du gestionnaire, mettant fin à son exécution.

En résumé, le joueur assure une interaction bidirectionnelle avec le gestionnaire, recevant des données initiales, envoyant des choix de cartes, recevant des mises à jour, mettant à jour le score, évaluant les conditions de fin de partie et affichant les résultats avant de se déconnecter.

Les joueurs robots

La transition du joueur humain au joueur robot a introduit plusieurs ajustements au niveau des structures de données, des fonctions et de la logique de jeu pour automatiser le processus de prise de décision.
Cependant , des structures et fonctions communes permettent d'établir une base partagée entre le joueur humain et le joueur robot, facilitant la gestion du jeu et de la communication avec le gestionnaire.
Voici les modifications et les ajouts spécifiques au joueur robot :

Fonctions :

- **peutPlacerCarteDansRangee :** Cette fonction détermine si une carte peut êtr placée dans une rangée spécifique en vérifiant si elle respecte les règles du jeu. Elle prend en compte la valeur numérique de la carte par rapport à la dernière carte de la rangée.

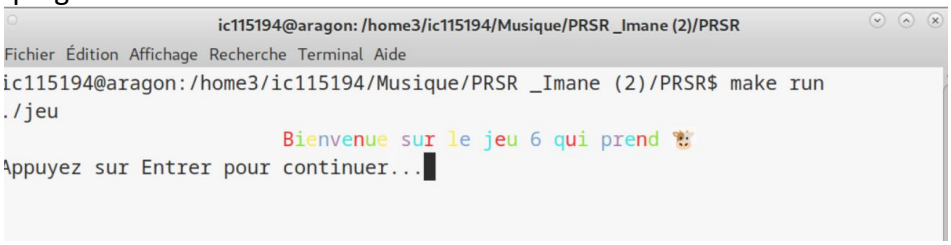
- **trouverIndiceRangeeCompatible** : Cette fonction trouve l'indice de la rangée la plus compatible pour une carte en analysant la différence de valeur numérique entre la carte et la dernière carte de chaque rangée. L'objectif est de minimiser les points de pénalité potentiels.
- **trouverPlusPetiteCartePlacable :** Cette fonction trouve la plus petite carte pouvant être placée dans une rangée. Elle est utilisée pour optimiser le choix de carte en minimisant les pertes potentielles.
- **trouverIndicePlusPetiteCartePlacable** : Cette fonction trouve l'indice de la plus petite carte pouvant être placée dans une rangée. Elle est utilisée en conjonction avec la fonction précédente pour déterminer où placer la carte de manière stratégique.
- **peutPlacerCarteDansUneRangee** :Cette fonction vérifie si une carte peut être placée dans l'une des rangées disponibles en utilisant les fonctions précédentes. Elle est utile pour décider où placer la carte de manière optimale.
- **resetCartesDejaChoisies** :Cette fonction réinitialise le tableau des cartes déjà choisies par le joueur robot. Elle est utilisée pour garantir que le robot ne choisit pas la même carte plusieurs fois lors de la même manche.

Stratégie et intelligence du robot :

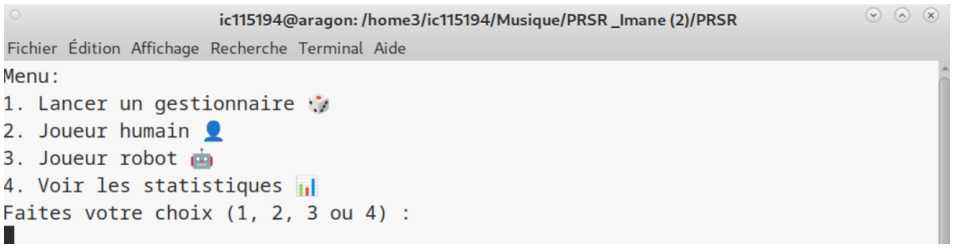
Le joueur robot démontre une intelligence stratégique en évaluant de manière dynamique ses cartes pour déterminer celles qui sont conformes aux règles du jeu.
Sa stratégie repose sur une analyse précise des écarts de valeur numérique entre ses cartes et la dernière carte de chaque rangée, utilisant des fonctions telles que "trouverIndiceRangeeCompatible" et "trouverPlusPetiteCartePlacable".
L'objectif principal est de minimiser les points de pénalité en choisissant judicieusement la rangée la plus compatible et la carte la moins numérique possible.
En cas d'absence d'options avantageuses, le robot limite les pertes en optant pour la carte de plus petite valeur numérique dans son paquet.
Sa capacité à s'ajuster aux mises à jour du gestionnaire, y compris les cartes des autres joueurs, témoigne d'une adaptabilité stratégique, conférant au joueur robot une compétence notable dans le contexte du jeu de cartes en réseau.

Le menu du jeu

Dans le processus Jeu.c, le script offre une expérience d'accueil créative en affichant un message de bienvenue coloré, chaque caractère étant présenté dans une couleur différente, le tout centré dans le terminal pour un impact visuel maximal.
L'ajout d'un Makefile facilite l'exécution du jeu en automatisant le processus de compilation et d'exécution du programme.

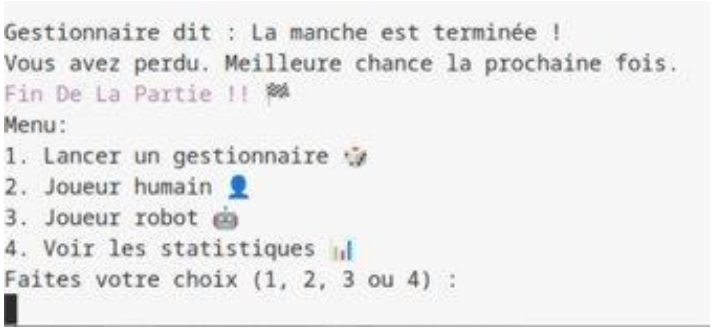


Une fois le message affiché, le script attend que l'utilisateur appuie sur la touche Entrée, permettant ainsi une interaction synchronisée. Ensuite, le script compile un autre fichier source C, vraisemblablement le menu principal du jeu, avant de nettoyer l'écran du terminal pour assurer une transition nette.



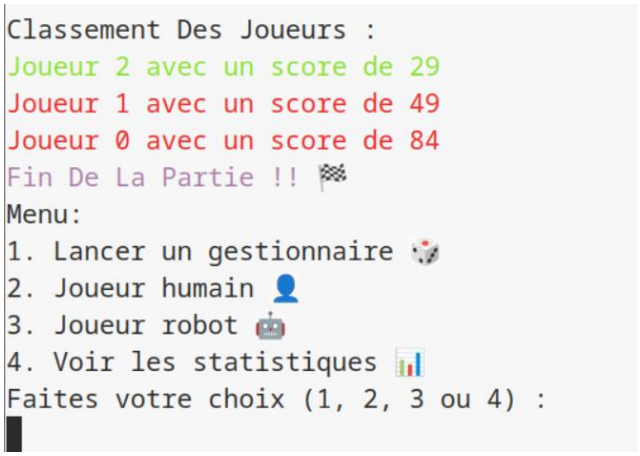
Enfin, le programme compilé du menu est exécuté, déclenchant ainsi l'interface principale du jeu. Concernant le processus Menu.c, il joue le rôle du menu principal du jeu et est exécuté à la suite du premier script. Ce menu propose à l'utilisateur diverses options, telles que le lancement d'un gestionnaire de jeu, le choix entre jouer en tant que joueur humain ou robot, et la consultation des statistiques de jeu.
Lorsqu'un utilisateur fait un choix, le script le valide avant de procéder. En fonction de cette sélection, le script prend des mesures appropriées, telles que compiler et lancer différents modules du jeu pour les joueurs humains ou robots, ou afficher les scores accumulés. Le menu est conçu pour fonctionner en continu, offrant ainsi à l'utilisateur la possibilité d'interagir de manière répétée avec les diverses options et fonctionnalités disponibles dans le jeu.

Chaque joueur est attribué une couleur spécifique pour ses cartes, créant ainsi une distinction visuelle entre les participants.
La représentation visuelle des cartes dans les rangées maintient la couleur attribuée à chaque joueur.
Lorsqu'un joueur choisit de jouer une carte, cette action est immédiatement visible pour son concurrent.
De plus, les scores individuels sont affichés en temps réel, offrant aux joueurs une vue claire de leur performance tout au long du jeu.



À la fin de chaque partie, le joueur perdant reçoit un message spécifique lui indiquant sa défaite. Ce message inclut également des encouragements pour la partie suivante afin de maintenir un esprit positif malgré la défaite.

D'autre part, le joueur gagnant reçoit des félicitations personnalisées, célébrant sa victoir. Le message de félicitations est accompagné du score atteint pendant cette partie particulière, renforçant ainsi le sentiment de réussite du joueur.



À la fin de chaque partie, le gestionnaire affiche le classement des joueurs , le score du joueur gagnant (en vert) et le ou les perdants en rouge .

Après la conclusion de chaque partie, le menu réapparaît, offrant aux joueurs la possibilité de jouer une nouvelle partie.

Ajustements suite à la présentation

Suite à vos remarques , nous avons :

- Donner la possibilité au gestionnaire de choisir le nombre de joueurs , dans le menu .
- Donner la possibilité au gestionnaire de choisir le nombre de manches , dans le menu .
- Permis au joueur de voir la carte jouer par son concurrent en temps réel .
- Ajouter le fichier makefile .

(Ces ajustements sont expliqués en détails ci-dessus) .