GIE Jimmy
LOGEROT Alexis

Projet 6 qui prend Systèmes et réseaux

Nous avons ajouté ce que vous nous aviez suggéré, c'est-à-dire pouvoir choisir le nombre de têtes de bœuf pour la partie pour ne pas être limité à une partie de 66 têtes de bœuf. Le makefile sous linux marche également, il y a aussi le fichier .exe sous windows. Pour générer le pdf sous linux vous devrez impérativement installer wkhtmltopdf avec :

Sudo apt install wkhtmltopdf

Sinon le pdf ne sera pas créé et rempli ; il y aura seulement les statistiques dans le fichier scores.log

Introduction
Objectif jeu du 6 qui prend, qui simule un jeu de cartes entre un ou plusieurs joueurs humains et un ou plusieurs robots. Le code source est organisé autour de structures de données et de fonctions permettant de gérer les cartes, de les distribuer, de les afficher, et de gérer la table du jeu.
L'implementation a été faite sous QtCreator, il est conseillé de lancer le jeu dans le terminal avec :

Structures de Données que nous avons utilisées.
Structure "Carte"

Une carte est représentée par une structure appelée "Carte". Chaque carte possède deux propriétés :

- **numero**: Représente le numéro de la carte.
- **valeur**: Représente la valeur attribuée à la carte.

Structure "Joueur"
Un joueur est modélisé par une structure nommée "Joueur". Cette structure contient :

- **nom**: Le nom du joueur.
- **main**: Un tableau de 10 cartes pour stocker les cartes de la main du joueur.
- **selection**: Un tableau pour stocker les cartes sélectionnées par le joueur.

Pour toutes les fonctions tout est détaillée en commentaire dans le code.

Sommaire

Réalisation

1. Processus

Nous avons utilisé qu'un seul processus qui gère tout le jeu.

2. Têtes de Bœuf

Pour les têtes de Bœuf nous avons choisi de respecter les valeurs de têtes de bœuf du jeu d'origine.

- Cartes multiples de 5 avec une valeur paire (10, 20, 30, 40, etc.) : valeur de tête de bœuf = 3
- Cartes multiples de 5 avec une valeur impaire (15, 25, 35, 45, etc.) : valeur de tête de bœuf = 2
- Cartes multiples de 11 sauf 55 (11, 22, 33, 44, etc.) : valeur de tête de bœuf = 5
- Carte 55 : valeur de tête de bœuf = 7
- Toutes les autres cartes : valeur de tête de bœuf = 1

C'est cette fonction qui nous permet de créer notre paquet de 104 avec chaque valeur de têtes de bœuf.

```
// creer un paquet de carte avec les valeur(tete de boeuf) associé a la carte
void creerPaquetDeCartes(Carte paquet[])
{
    for (int i = 0; i < 104; i++)
    {
        // carte 10,20,30,40....
        paquet[i].numero = i + 1;
        if ((i + 1) % 5 == 0)
        {
            if ((i + 1) % 2 == 0)
            {
                paquet[i].valeur = 3;
            }
            else
            {
                // carte 55
                if (i == 54)
                {
                    paquet[i].valeur = 7;
                    // carte 15,25,35,45...
                }
                else
                {
                    paquet[i].valeur = 2;
                }
            }
            //carte 11,22,33,44...
        }
        else if ((i + 1) % 11 == 0)
        {
            paquet[i].valeur = 5;
            //toutes les autres cartes
        }
        else
        {
            paquet[i].valeur = 1;
        }
    }
}
```

3. Max Joueurs

Pour le maximum de joueurs, nous avons également choisi de respecter les règles du jeu de base, c'est-à-dire de pouvoir jouer à 10 maximum. Nous pouvons jouer avec des joueurs réels, c'est-à-dire des joueurs qui rentrent leur pseudo en début de partie et qui sélectionnent l'indice de la carte qu'ils veulent jouer.

On peut également jouer avec des robots qui sélectionnent une carte aléatoire dans leurs mains ou avec des robots intelligents qui sélectionnent une carte spécifique en fonction du jeu (voir robot intelligent).

Pour l'implémentation, on demande à l'utilisateur avec combien de joueurs il souhaite lancer la partie. Si l'utilisateur met un nombre supérieur à 10, un message d'erreur apparait et le programme s'arrête. Une fois le nombre de joueurs choisi. Par exemple : Pour 8 joueurs on va lui demander le nombre de joueurs réels (humains) sur 8. Ici, si l'utilisateur choisit 3 il y aura donc 3 joueurs humains et 5 robots, S'il avait choisie 0 il n'y aurait que des robots et si il avait mis une valeur supérieure à 8, un message d'erreur serait apparu et le programme se serait arrêté.

```
int nombreJoueursTotal, nombreJoueursReels, difficulte;

printf("Entrez le nombre de joueurs total (maximum 10) : ");
scanf("%d", &nombreJoueursTotal);     ⚠Call to function 'scanf' is insecure as it does not prov

// Vérifier la limite pour le nombre total de joueurs
if(nombreJoueursTotal > MAX_JOUEURS_TOTAL)
{
    printf("Nombre total de joueurs depasse la limite de 10.\n");
    return;
}


printf("Entrez le nombre de joueurs reels (humains) parmi %d joueurs : ", nombreJoueursTotal);
scanf("%d", &nombreJoueursReels);     ⚠Call to function 'scanf' is insecure as it does not prov

// Vérifier la limite pour le nombre de joueurs réels
if(nombreJoueursReels > nombreJoueursTotal)
{
    printf("Nombre de joueurs reels dépasse le nombre total de joueurs.\n");
    return;
}
```

Dans l'exemple ci-dessous l'utilisateur a choisi 2 joueurs avec 0 joueur réel, soit 2 robots.

```
jimmy@LinuxDebian12:/media/sf_partageVB/ProjetSR1/ProjetSR$ ./6quiprendLinux
Entrez le nombre de joueurs total (maximum 10) : 2
Entrez le nombre de joueurs reels (humains) parmi 2 joueurs : 0
```

4. Affichage

Pour le bon déroulement du jeu et sa compréhension nous avons choisi d'afficher plusieurs choses comme :

-La création du paquet de 104 cartes au début de chaque manche.

-Le mélange de ce même paquet de cartes.

-La main de chaque joueur (on peut bien évidement cacher leur main, mais c'est pour voir si le jeu se déroule correctement) :

-la carte choisie par chaque joueur pour le tour

-Le plateau de cartes, les scores de chaque joueur pour le tour et pourquoi tel ou tel joueur a obtenu des têtes de bœuf. Par exemple : robot1 a obtenu 2 têtes de bœuf car sa carte est inférieure à toutes les autres.

Dans l'exemple ci-dessous, on a une partie entre 2 robots avec chacun leur main avec une carte à la valeur 0, car elle a déjà été posée au tour 1. Ensuite la carte sélectionnée à ce tour pour chaque joueur est affichée. Les cartes sont ensuite posées sur le plateau de jeu et leur score pour le tour est ensuite affiché.

```
TOUR : 2
Main de Robot1 avant selection :
Main du joueur Robot1 :
0 :(57, 1)
1 :(93, 1)
2 :(79, 1)
3 :(38, 1)
4 :(63, 1)
5 :(0, 0)
6 :(52, 1)
7 :(1, 1)
8 :(59, 1)
9 :(47, 1)
Main de Robot2 avant selection :
Main du joueur Robot2 :
0 :(28, 1)
1 :(20, 3)
2 :(50, 3)
3 :(0, 0)
4 :(95, 2)
5 :(90, 3)
6 :(68, 1)
7 :(99, 5)
8 :(42, 1)
9 :(94, 1)
Carte sélectionnee pour Robot1 :
(38, 1)
Carte sélectionnee pour Robot2 :
(42, 1)
Liste des joueurs triee pour le tour 2 :
Joueur 1 : Robot1
Joueur 2 : Robot2
Carte selectionnée par le(s) joueur(s) et le(s) robot(s)(38, 1) (42, 1)

Le joueur Robot1 a recuperé 1 têtes de bœuf car aucune ligne n'avait une valeur
plus petit que sa carte.

Le joueur Robot1 a un total de 1 têtes de bœuf pour ce tour.
Le joueur Robot2 a un total de 0 têtes de bœuf pour ce tour.
(38, 1) (42, 1) (0, 0) (0, 0) (0, 0) (0, 0)
(19, 1) (54, 1) (69, 1) (0, 0) (0, 0) (0, 0)
(100, 3) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0)
(76, 1) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0)
FIN DU TOUR : 2
```

5. Statistiques

Pour les statistiques nous créons un fichier scores.log, que l'on convertit en fichier pdf à la fin de la partie, celui-ci comprend les scores de chaque joueur à chaque fin de manche, ainsi que leur classement à la fin de la partie.

La fonction sauvegarderScores nous permet d'enregistrer chaque score de manche avec un script AWK, et la fonction ConvertToPDF nous permet de sauvegarder le fichier scores.log en pdf à la fin de la partie.

Pour que converToPDF marche chez vous sous linux vous devez installer wkhtmltopdf avec la commande :

Sudo apt install wkhtmltopdf

```c
void sauvegarderScores(int numeroManche, char nomJoueur[], int score)
{
    char commande[200]; // Augmentons la taille de la chaîne de commande pour plus de sécurité

    // Utilisation de sprintf pour générer la commande complète avec les valeurs des variables
    sprintf(commande, "awk -v nm=%d -v nj='%s' -v sc=%d 'BEGIN {print \"MANCHE : \" nm \", JOUEUR : \" nj \", SCORE : \" sc}' >> scores.log", numeroManche, nomJoueur, score);

    // Exécution de la commande via system()
    system(commande);
}

//convertie le fichier scores.log en fichier pdf grace a la librairie wkhtmltopdf qui doit etre installer sur l ordi avec-> sudo apt install wkhtmltopdf
void convertToPDF()
{
    system("wkhtmltopdf scores.log scores.pdf");
}
```

```
MANCHE : 1, JOUEUR : Robot1, SCORE : 16
MANCHE : 1, JOUEUR : Robot2, SCORE : 0
MANCHE : 2, JOUEUR : Robot1, SCORE : 26
MANCHE : 2, JOUEUR : Robot2, SCORE : 9
MANCHE : 3, JOUEUR : Robot1, SCORE : 39
MANCHE : 3, JOUEUR : Robot2, SCORE : 15
MANCHE : 4, JOUEUR : Robot1, SCORE : 50
MANCHE : 4, JOUEUR : Robot2, SCORE : 15
MANCHE : 5, JOUEUR : Robot1, SCORE : 56
MANCHE : 5, JOUEUR : Robot2, SCORE : 29
MANCHE : 6, JOUEUR : Robot1, SCORE : 69
MANCHE : 6, JOUEUR : Robot2, SCORE : 44
Classement des joueurs :
1. Joueur Robot2 - Score : 44
2. Joueur Robot1 - Score : 69
```

Règles du Jeu

1. Placement de Cartes

Le placement des cartes est fait de manière automatique grâce à notre fonction ajouterCartesAuTable qui prend en paramètre le plateau de cartes, la liste de cartes triées dans l'ordre croissant (une carte par joueur), la liste de joueurs triée dans l'ordre de leur carte, ainsi que le nombre de joueurs.

```c
void ajouterCartesAuTable(Carte table[][6], Carte cartes[], Joueur joueurs[10], int taille)
```

2. Mise à Jour des Listes

Pour la mise à jour des listes, nous en avons 3 différentes :

-Une qui nous affiche notre paquet de 104 cartes à la fin de chaque manche avec les cartes restantes. Par exemple : si l'utilisateur sélectionne 3 joueurs pour la partie, 4 cartes seront posées sur la table et 10 cartes seront distribuées par joueur soit 104-10-10-10-4=70. A la fin de la manche, le programme nous affiche alors les 70 cartes non utilisées. S'il y a une nouvelle manche ont recréé un paquet de 104 cartes et on recommence.

-Notre liste de cartes sélectionnées soit une par joueur, si l'on reprend l'exemple des 3 joueurs au-dessus il y aura 3 cartes dans cette liste. `Carte cartesSelectionnees[nombreJoueursTotal];`

Cette liste est triée dans l'ordre croissant et utilisée dans notre fonction ajouterCartesAuTable. Une fois les cartes utilisées dans le jeu, la liste est remise à 0 à chaque fin de tour. Grâce à `memset(cartesSelectionnees, 0, sizeof(cartesSelectionnees));`,cela nous évitons de stocker toutes les cartes du jeu dans cette liste et d'avoir une erreur au moment où elles sont posées.

3. 6ème Carte

La 6eme carte est posée automatiquement grâce à notre fonction ajouterCartesAuTable grâce à ce morceau de code :

```c
if (ligneAEnlever != -1)
{
    // Vérifie si la cinquième carte a la différence minimale
    int differenceCinquiemeCarte = cartes[i].numero - table[ligneAEnlever][4].numero;
    if (differenceCinquiemeCarte == differenceMin)
    {
        // Supprime la ligne en ajoutant des zéros
        for (int k = 0; k < 6; k++)
        {
            //printf("valeur carte a %d\n", table[ligneAEnlever][k].valeur);
            p1 =p1+table[ligneAEnlever][k].valeur;
            table[ligneAEnlever][k].numero = 0;
            table[ligneAEnlever][k].valeur = 0;
        }
        printf("Le joueur %s a recuperé %d têtes de bœuf car sa carte avait la difference minimum avec une ligne possedant deja 5 cartes\n\n", joueurs[i].nom, p1);
    }
}
if (ligneLaPlusProche != -1)
{
    for (int k = 0; k < 6; k++)
    {
        if (table[ligneLaPlusProche][k].numero == 0)
        {
            table[ligneLaPlusProche][k] = cartes[i]; // Place la carte sélectionnée dans la première case vide de la ligne la plus proche
            break; // Sort de la boucle
        }
    }
}
```

Précédemment le code a vérifié si la dernière carte a la valeur minimum entre la carte de la liste `Carte cartesSelectionnees[nombreJoueursTotal];` pour chaque carte la plus à droite sur chaque ligne, si c'est le cas notre code ci-dessus vérifie si cette ligne comporte 5 cartes, si c'est le cas ces 5 cartes sont alors mises à 0 et la carte de la liste carteSelectionnees est alors mise en première position.

4. La Plus Petite Carte

Pour la plus petite carte, toujours dans la fonction ajouterCartesAuTable, c'est quasiment la même chose que pour la 6eme carte. Chaque carte la plus à droite de chaque ligne est comparée avec la carteSelectionnees. Si la différence minimum est négative pour la carte, alors la ligne est supprimée et la carteSelectionnees est mise en première position.

5. Conditions d'Arrêt

Pour les conditions d'arrêt, nous avons laissé l'utilisateur choisir le nombre de têtes de bœuf qu'il faut avoir pour que la partie s'arrête. C'est à dire qu'en début de partie l'utilisateur entre un nombre, s'il rentre 70 le jeu s'arrêtera lorsqu'un joueur aura atteint 70 têtes de bœuf ou plus.

```
jimmy@LinuxDebian12:/media/sf_partageVB/ProjetSR1/ProjetSR$ ./6quiprendLinux
Entrez le nombre de joueurs total (maximum 10) : 2
Entrez le nombre de joueurs reels (humains) parmi 2 joueurs : 0
Avec combien de tetes de beoufs voulez vous jouer ?
70
Contre quelle dificultee de robot voulez vous jouer ?
Robot aleatoire tapez: 1.
Robot intelligent tapez:2.
```

Bien évidemment, l'utilisateur ne peut pas choisir de valeur en dessous de 0.

```
Classement des joueurs :
1. Joueur Robot2 - Score : 31
2. Joueur Robot1 - Score : 73
```

Si on va à la fin de la partie on peut bien voir que le jeu c'est arrêter car le joueur Robot1 a atteint 70 ou plus et qu'il est le dernier du classement.

De plus, pour vérifier si l'on est dans la condition d'arrêt, une fonction trouveerScoreMax est utilisée pour vérifier à chaque fin de tour et de manche si le joueur qui a le plus de têtes de bœuf a atteint la limite donnée par l'utilisateur en début de partie.

```c
// Fonction pour trouver le score maximum parmi les joueurs
int trouverScoreMax(Joueur joueurs[], int nombreJoueursTotal)
{
    int scoreMax = -1; // Initialisation avec une valeur basse possible

    for (int g = 0; g < nombreJoueursTotal; g++)
    {
        if (joueurs[g].scores > scoreMax)
        {
            scoreMax = joueurs[g].scores;
        }
    }
    return scoreMax;
}
```

Cette fonction prend en paramètre la liste de joueurs et combien ils sont. Le score de chaque joueur et ensuite vérifie et stockée dans scoreMax. S'il a plus de têtes de bœuf que le joueur précédant. Le scoreMax de ce joueur est ensuite retournée, il est ensuite vérifié dans les deux boucles while de la fonction jeu, une pour les tours, une pour les manches en fonction du nombres de têtes de bœuf qu'a rentré l' utilisateur en début de partie.

6. Score

Les scores sont calculés avant chaque suppression de ligne et associés au joueur qui a posé sa carte.

```c
//printf("valeur carte a %d\n", table[ligneAEnlever][k].valeur);
p1 =p1+table[ligneAEnlever][k].valeur;
table[ligneAEnlever][k].numero = 0;
table[ligneAEnlever][k].valeur = 0;
```

La valeur de la ligne est calculée avant d'etre mise à 0, et est ensuite associée au joueur à la fin du tour.

P1 pour les lignes contenant 5 cartes et P pour les lignes qui on une valeur superieure à la carteSelectionnees

```c
// Ajout des scores pour ce tour au score total du joueur
int totalScore = p + p1;
joueurs[i].scores += totalScore;

// Affichage du score total pour ce tour
printf("Le joueur %s a un total de %d têtes de bœuf pour ce tour.\n", joueurs[i].nom, joueurs[i].scores);
```

Les scores sont affichés à chaque fin de tour et fin de manche.

```
Le joueur Robot1 a recuperé 8 têtes de bœuf car sa carte avait la difference min
imum avec une ligne possedant deja 5 cartes

Le joueur Robot1 a un total de 73 têtes de bœuf pour ce tour.
Le joueur Robot2 a recuperé 8 têtes de bœuf car sa carte avait la difference min
imum avec une ligne possedant deja 5 cartes

Le joueur Robot2 a un total de 31 têtes de bœuf pour ce tour.
(16, 1) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0)
(90, 3) (91, 1) (98, 1) (0, 0) (0, 0) (0, 0)
(97, 1) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0)
(53, 1) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0)
FIN DU TOUR : 8

Le joueur Robot1 a un total de 73 têtes de bœuf pour la fin de cette manche.
Le joueur Robot2 a un total de 31 têtes de bœuf pour la fin de cette manche.
Classement des joueurs :
1. Joueur Robot2 - Score : 31
2. Joueur Robot1 - Score : 73
```

Fonctionnalités Additionnelles

2. Robot Intelligent

Pour jouer contre un robot intelligent, une option se présente à l'utilisateur en début de partie. Une pour jouer contre le robot qui sélectionne aléatoirement dans sa main et une autre option pour le robot intelligent qui lui vérifie la différence avec les 10 cartes qu'il à dans sa main et la carte la plus à droite de chaque ligne.

```
int selectionnerCarteRobot(Carte table[][6], Carte main[])
```

Ceci est fait grâce à notre fonction selectionnerCarteRobot qui prend en paramètre le plateau de jeu et la main du joueur, c'est à dire ses 10 cartes.

La fonction compare une à une chaque carte de la main du joueur avec les 4 cartes sur la table. Elle sauvegarde dans une liste la valeur minimum de chaque carte avec l'indice de la main du joueur, une fois fait, avec toutes les cartes, la différence minimum entre chaque carte est triée dans l'ordre croissant et l'indice de la carte avec la plus petite différence est retournée. (Voir code dans la classe joueur).

Compilation et Lisibilité

1. Compilation

Pour la compilation, nous utilisons l'outil make, la commande : make a juste à être exécuté, cela créera un fichier 6quiprendLinux, qui pourra être exécuté à son tour avec la commande ./ 6quiprendLinux

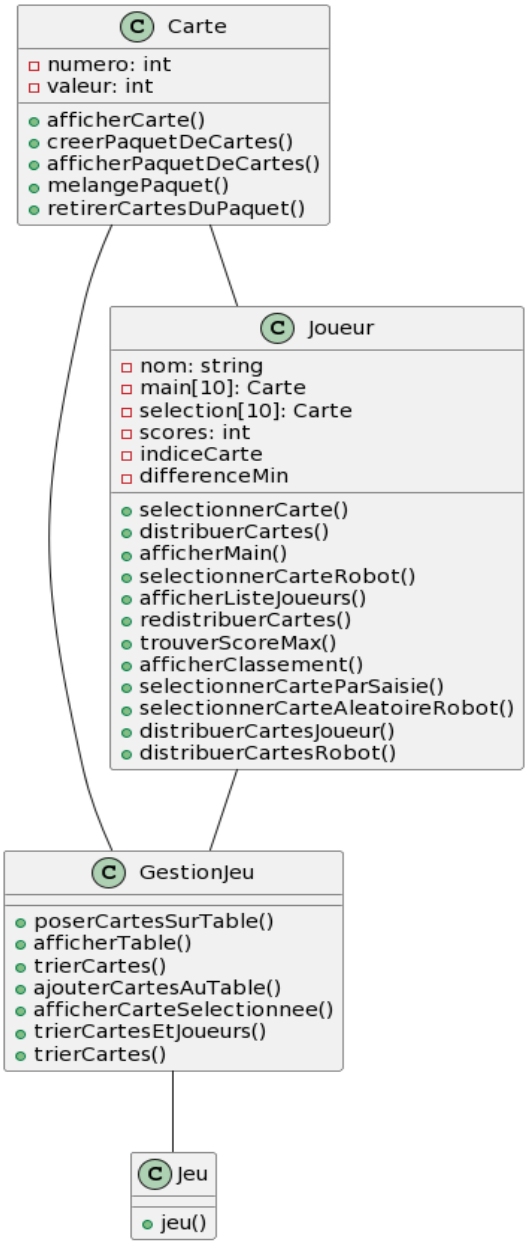
```
CC = gcc
CFLAGS = -Wall -m64
LIBS = -lm

SRCS = jeu.c gestionjeu.c joueur.c carte.c main.c
OBJS = $(SRCS:.c=.o)
EXEC = 6quiprendLinux

all: $(EXEC)

$(EXEC): $(OBJS)
	$(CC) $(CFLAGS) -o $@ $(OBJS) $(LIBS)

%.o: %.c
	$(CC) $(CFLAGS) -c $< -o $@

clean:
	rm -f $(EXEC) $(OBJS)
```

Le Makefile permet de compiler un ensemble de fichiers source C en utilisant **gcc**, d'appliquer des options spécifiques lors de la compilation et de créer un exécutable appelé **6quiprendLinux**. La règle **clean** permet de supprimer les fichiers générés lors de la compilation.

2. Lisibilité

Nous avons décidé d'indenter le code avec l'indentation de type **indentation horizontale.**

3. Diagramme


Carte
numero: int
valeur: int
afficherCarte()
creerPaquetDeCartes()
afficherPaquetDeCartes()
melangePaquet()
retirerCartesDuPaquet()
Joueur
nom: string
main[10]: Carte
selection[10]: Carte
scores: int
indiceCarte
differenceMin
selectionnerCarte()
distribuerCartes()
afficherMain()
selectionnerCarteRobot()
afficherListeJoueurs()
redistribuerCartes()
trouverScoreMax()
afficherClassement()
selectionnerCarteParSaisie()
selectionnerCarteAleatoireRobot()
distribuerCartesJoueur()
distribuerCartesRobot()
GestionJeu
poserCartesSurTable()
afficherTable()
trierCartes()
ajouterCartesAuTable()
afficherCarteSelectionnee()
trierCartesEtJoueurs()
trierCartes()
Jeu
jeu()