Revue de projet

Sommaire

Introduction

Au cours de notre projet nous avons du implémenter en C le jeu des petits chevaux en suivant les règles énoncées sur la page Wikipédia de ce jeu tout en respectant les consignes exigées.

Jeu.h

Les structures :

Dans une partie nous auront besoin de plusieurs joueurs qui auront accès à plusieurs chevaux.

Chacun d'eux pourra être représenté par des variables basées sur le même modèle.

J'ai tout d'abord créé la structure joueur.

```
struct joueur  {
	char *pseudo;
	int ordi;
	int actif;
	int chactif;
	Cheval ecurie[4];
	int gagnant;
};

typedef struct joueur Joueur;
```

Pour les besoins du code nous aurons besoin de connaître les informations suivantes :

Son pseudonyme

Si il est un ordinateur ou joueur réel.

Si il est actif.

Combien de chevaux sont sortis de l'écurie

Si le joueur a gagné

Nous aurons aussi besoin d'un tableau de structure pour représenter les quatre chevaux de chaque joueur.

```
struct cheval {
	int depart;
	int echelle;
	int emplacement;
	int tour;
} ;

typedef struct cheval Cheval;
```

La structure cheval est composée de la case départ, de la variable échelle qui représente la prochaine marche de l'échelle, de l'emplacement actuel et du nombre de tour fait. Cette dernière sera utile pour tous les joueurs sauf le premier.

Jeu.c

Pour créer nos fonctions nous auront besoin des librairies suivantes :

```
#include <stdio.h>
#include <stdlib.h>
#include "jeu.h"
#include <time.h>

#include<windows.h>
```

L'initialisation :

Tout d'abord nous auront besoin de savoir le nombre de joueur et d'ordinateur.

```
int nbjoueur = 0;
	int nbordi = 0;
	printf("nombre de joueur (0 a 4): ");
	scanf_s("%d", &nbjoueur);
	if (nbjoueur > 4) {
		printf("le nombre de joueur est trop grand il y en aura donc 4 \n");
		nbjoueur = 4;
	}
```

```
	else {
		printf("nombre d'ordinateur voulue (0 a %d): ",4-nbjoueur);
		scanf_s("%d", &nbordi);
		if (nbjoueur + nbordi > 4) {
			nbordi = 4 - nbjoueur;
			printf("nombre d'ordinateur trop grand il n'y aura que %d ordinateur
\n", nbordi);
		}
	}
```

Puis nous allons initialiser tous les participants et joueurs.

```
for (int i = 0; i < nbjoueur; i++) {
	printf("le joueur s'appel %s ", groupe[i].pseudo);
	groupe[i].ordi = 0;
	groupe[i].actif = 0;
	groupe[i].chactif = 0;
	groupe[i].gagnant = 0;
	for (int k = 0; k < 4; k++) {
		groupe[i].ecurie[k].depart = 1+(i*14);
		groupe[i].ecurie[k].echelle = 0;
		groupe[i].ecurie[k].emplacement = 0;
		groupe[i].ecurie[k].tour = 0;
	}
}


for (int j = nbjoueur; j < (nbjoueur + nbordi); j++) {
	groupe[j].ordi = 1;
	groupe[j].actif = 0;
	groupe[j].chactif = 0;
	groupe[j].gagnant = 0;
	for (int k = 0; k < 4; k++) {
		groupe[j].ecurie[k].depart = 1+(j*14);
		groupe[j].ecurie[k].echelle = 0;
		groupe[j].ecurie[k].emplacement = 0;
		groupe[j].ecurie[k].tour = 0;
	}
}
printf("le jeu est initialise \n");
return nbordi + nbjoueur;

}
```

Le dé :

Pour le dé je suis allée chercher la fonction sur le site « stack overflow » qui renvoie un nombre aléatoire. Auquel j'ai rajouté un délai de 900 ms. j'ai remarqué que la fonction renvoyé une chiffre qui change à chaque seconde et puisqu'un dé peux donner deux fois de suite le même nombre 1 s aurait était trop long.

```
int de() {
	srand(time(NULL));
	int d = (rand() % 6) + 1;
	Sleep(900);
	return d;

}
```

Sortir les chevaux :

Dans les règle de Wikipédia il est écrit qu'un joueur ne peux commencer qu'en ayant obtenu un 6 au lancé de dé, par ailleurs au cours du jeu, si le joueur obtient de nouveau un 6 alors il a le choix de sortir un cheval. C'est pourquoi j'ai utilisé la même fonction qui a était nommée « commence ».

```
void commence(Joueur* groupe, int i, int d) {
	if (d == 6) {
		groupe[i].actif = 1;
		groupe[i].ecurie[groupe[i].chactif].emplacement =
groupe[i].ecurie[groupe[i].chactif].depart;
		groupe[i].chactif += 1;
		printf("le joueur %d sort le cheval %d \n", i+1, groupe[i].chactif);

	}
}
```

Le déplacement et échelle :

Le déplacement est la fonction qui m'a demandé le plus de temps et qui est également la plus longue. Si le joueur n'est pas un ordinateur on va commencer par lui demander quel cheval veut-il choisir ? Puis le déplacer en fonction du dé.

```
int c = 1;
	int done = 0;
	if (groupe[i].chactif >= 1) {
		printf("quel cheval voulez-vous bouger? ");
		scanf_s("%d", &c);
		while (c != 1 && c != 2 && c != 3 && c != 4 || c > groupe[i].chactif) {
			printf("veuillez choisir une nombre convenable ");
			scanf_s("%d", &c);
		}
	}
```

Il avance de son emplacement plus la valeur du dé s'il est inférieur à l'emplacement de depart-1

```
groupe[i].ecurie[c - 1].emplacement += d;
```

recule si il est supérieur

```
groupe[i].ecurie[c - 1].emplacement = (groupe[i].ecurie[c - 1].depart - 1) - recule;
```

et si il est égal alors il a l'oportunité de monter sur l'echelle

```
groupe[i].ecurie[c - 1].emplacement += d;

if (groupe[i].ecurie[c - 1].echelle == 0) {
	groupe[i].ecurie[c - 1].echelle = 1;
}
if (groupe[i].ecurie[c - 1].echelle == d || groupe[i].ecurie[c - 1].echelle == 7 && d == 6) {
	groupe[i].ecurie[c - 1].echelle += 1;

}
```

Pour que le cheval monte sur l'échelle il faut que la valeur du dé soit égale à la variable « échelle » situé dans la structure.

Il y a un cas particulier pour le premier joueur puisque c'est le seul qui a un emplacement de départ inférieur à celui d'arrivée.

Les ordinateurs, eux, ont un fonctionnement similaire a une exception prêt, si le déplacement entraine que le cheval va « manger » un de ses congénères d'écurie, alors il tente de déplacer le prochain cheval.

Une fois le déplacement effectué, nous allons appeler la fonction « manger » puis afficher l'état des chevaux du joueur en cours.

Le retour à l'écurie :

La fonction qui permet de ramener à l'écurie un cheval si un autre arrive sur la même case est la fonction « manger » qui vérifie chaque emplacement pour savoir si il est diffèrent du dernier qui a était déplacé.

```
void manger(Joueur* groupe, int i, int j, int nbparticipant) {
	if (groupe[i].ecurie[j].echelle == 0) {
		for (int l = 0; l < nbparticipant; l++) {
			for (int k = 0; k < 4; k++) {
				if (groupe[i].ecurie[j].emplacement == 
groupe[l].ecurie[k].emplacement && groupe[l].ecurie[k].echelle == 0) {
					groupe[l].chactif -= 1;
					int a = groupe[l].ecurie[k].emplacement;
					for (int m = k; m < 3; m++) {
						groupe[l].ecurie[k] = groupe[l].ecurie[k + 1];
					}
					groupe[l].ecurie[3].emplacement = 
groupe[l].ecurie[k].depart;
					groupe[l].ecurie[3].tour = 0;
				}
			}
		}
	}
}
```

Malheureusement je n'ai pas réussi à faire marcher cette fonction donc je l'ai mis en commentaire dans le code.

Main.c

Assemblage :

L'assemblage commence par le menu qui propose de commencer une partie ou de quitter le jeu

```
int menu;
printf("voulez-vous: \ncommencer une parti(1) \nquitter le jeu(2) \n");
scanf_s("%d",&menu);
if (menu == 1) {
	Joueur groupe[4];
	int i = 0;
	int d;
}
```

On initialise le jeu

```
int nbparticipant;
nbparticipant = init_jeu(&groupe);
```

Pour chaque participant on lance le dé

```
for (int i = 0; i < nbparticipant; i++) {
	int recommence = 0;
	printf("joueur %d pour lancer le dé touché n'importe quelle touche \n", i + 1);
	system("pause");
	d = de();
	printf("le joueur %d a eu %d au lance du de \n", i + 1, d);
```

Si il est égal à 6 le choix est laissé de sortir un cheval ou d'avancer puis le joueur relance le dé et reprend le cours de sa partie.

```
if (groupe[i].actif == 1) {
	if (d == 6 && groupe[i].chactif>0) {
	int choix;
	recommence = 1;
	if (/*groupe[i].ordi == 1 &&*/ groupe[i].chactif<4) {
		choix = 1;
		commence(&groupe, i, d);
	}
	else {
		if(groupe[i].chactif < 4){
			printf("voulez-vous sortir(1) un cheval ou avancer(2): ");
			scanf_s("%d", &choix);
			while (choix != 1 && choix != 2) {
				printf("veuillez choisir une valeur convenable");
				scanf_s("%d", &choix);
			}
			if (choix == 1) {
				commence(&groupe, i, d);
			}
			else {
				deplacement(&groupe, i, d, nbparticipant);
			}
		}
		else {
			printf("le joueur %d ne peux que avance \n", i + 1);
			deplacement(&groupe, i, d, nbparticipant);

		}

	}

}
```

```
if (recommence == 1) {
	d = de();
	printf("le joueur %d a eu %d au lance du de \n", i + 1, d);
	recommence == 0;
}
if (groupe[i].gagnant ==0) {
	deplacement(&groupe, i, d, nbparticipant);
}
```

Sinon il avance directement

```
if (groupe[i].gagnant ==0) {
	deplacement(&groupe, i, d, nbparticipant);
}
```

On regarde tous les joueurs, si l'un d'eux est à l'échelle 8 autrement dit si il est allé jusqu'en haut de l'échelle et a eu à nouveau un 6 alors il a gagné et le jeu s'arrête.

```
for (int j = 0; j < groupe[i].chactif; j++) {
	if (groupe[i].ecurie[j].echelle == 8) {
		printf("le joueur %d a gagne avec le cheval numero %d \n", i + 1, j + 1);
		return 0;
	}
if (groupe[i].actif == 0) {
	commence(&groupe, i, d);
}
else {
	return 0;
}
system("pause");
return 0;
```

Conclusion

Malheureusement je n'ai pas réussi à faire fonctionner ma fonction « manger » qui ramène les chevaux à l'écurie. Si j'avais eu plus de temps j'aurais beaucoup aimé faire fonction mon code dans son l'intégralité, rajouter un plateau qui montre l'évolution de la partie et rendre les ordis plus intelligents.

Surement à cause d'un manque d'organisation ce projet m'a pris énormément de temps mais m'a appris une méthode de travail en plus d'approfondir mes connaissance en C.