Réalisation de Projet
Projet: 1000 Bornes

Etienne PENAULT
Julien AUCLAIR
Lina TLEMÇANI

Semestre 4

Sommaire

Introduction

Le *1000 Bornes* est un jeu de société inventé dans les années 50'. Son but est simple, plusieurs joueurs ont chacun une main avec 6 cartes; le premier arrivé à 1000 kilomètres, ni plus ni moins, gagne la partie. Tout au long du jeu, chaque joueur peut appliquer des malus aux autres joueurs afin de les ralentir et donc arriver à atteindre les milles bornes avant eux !

Début du Projet

Pour commencer, au début du projet, nous avons réflechis comment représenter au mieux le jeu du milles bornes par des structures de données. Nous avons donc débattu puis, sommes arrivés à se mettre d'accord sur les structures que nous utiliserions tout au long du projet. Notre *1000 Bornes* est donc composé:

- de Cartes

```
struct Carte{
    int type; //Type de la carte
    int sous_type; // Sous type de la carte
};
typedef struct Carte carte;
```

- de Joueurs

```
struct Joueur{
    int demarre;
    int distance;
    int tour;
    int deuxcent;
    carte malus[NB_CARTE_MALUS]; //stock les malus
    carte bonus[NB_CARTE_BONUS]; //stock les bonus
    carte main[NB_CARTE];
};
```

```
typedef struct Joueur joueur;

struct Tab_Joueurs{
/*Tableau comportant nos joueurs*/
    int n;
    joueur * nos_joueurs;
};
typedef struct Tab_Joueurs * tab_joueurs;
```

- d'une Pioche

```
struct Jeu{
/*Tableau comportant nos cartes*/
    int n;
    carte * deck;
};
typedef struct Jeu * jeu;
```

Nous pouvous noter qu'une carte est définie principalement par deux attributs, son type et son sous type (assigné par des entiers). Voici comment nous avons pensé la classification des cartes:

- Cartes KM -0-:
  - 25 (10 unités) -0-
  - 50 (10 unités) -1-
  - 75 (10 unités) -2-
  - 100 (12 unités) -3-
  - 200 (4 unités) -4-
- Cartes bonus -1-:
  - As du Volant -0-
  - Camion citerne -1-
  - Increvable -2-

  - Prioritaire -3-

- Cartes Attaque -2-:
  - Accident (3 unités) -0-
  - Panne d'essence (3 unités) -1-
  - Crevaison (3 unités) -2-
  - Limite de Vitesse (4 unités) -3-
  - Feu rouge (5 unités) -4-
- Cartes Parades -3-:
  - Réparation (6 unités) -0-
  - Essence (6 unités) -1-
  - Roue de secours (6 unités) -2-
  - Fin de limitation (6 unités) -3-
  - Feu vert (14 unités) -4-

Une fois cela fait, nous nous sommes fait une boîte à outils afin de nous faciliter la tâche pour manipuler nos structures de données, comme des *Push* ou des *Pop*, ou encore des fonctions pour nous faciliter la tâche coté mémoire.

Fonctionnement

Lors du début d'une partie, nous commencons par définir une taille de 106 cartes à notres jeu afin de l'allouer et de le remplir. Une fois cela fait, on demande de saisir à l'utilisateur le nombre de joueurs (entre 2 & 8). Enfin, on mélange le jeu par la fonction "shuffle_jeu()", on initialise nos joueurs, puis nous distribuons nos cartes mélangées aux différents joueurs. La partie commence enfin !

Le *1000 Bornes* repose sur le tour par tour (que nous simulerons avec une boucle while, avec comme condition d'arret, que personne n'a encore gagné),

ce qui veut dire qu'à chaque tour, nous incrémentons l'index du joueur actuel, hormis quelques exceptions (par exemple, un joueur peut rejouer après avoir un bonus). Le programme considère qu'un joueur à jouer lorqu'il défosse une carte, (et en pioche une automatiquement après).
La fonction "jouer_carte()" est la fonction sur laquelle le jeu repose. C'est elle qui va permettre de détecter le type de carte que nous voulons jouer, qui retournera des erreurs en cas d'impossibilité de jouer, et qui permettra d'appliquer des malus aux autres participants. Si cette fonction retourne une erreur, un message spécifique à l'erreur sera retransmis au joueur afin de lui indiquer, et il sera reinvité à jouer.

Selon le type de la carte, le joueur devra cibler un autre joueur si la carte jouée est un malus. Nous pouvons aussi noter que si un joueur possède un bonus non joué dans sa main, et que le malus qui lui est relié est joué sur ce joueur, il y a ce qu'on appel un *Coup Fourré*. La fonction "coup_fourre()" gérera ce cas précédent, fera automatiquement jouer le bonus en question, et incrémentera le score de joueur de 400 bornes.
Une fois un joueur arrivé à 1000 bornes, on sort de la boucle puis on affiche le score de tous les joueurs.

Factorisation

Pour améliorer l'efficacité et la lisibilité du code du *1000 Bornes*, nous avons factorisé plusieurs fonctions.

Dans la fonction "affichage()" en plus des redondances, les lignes de codes n'étaient pas trés lisibles, en effet pour l'affichage des informations: joueurs, distances, bonus, attaques et parades, le nom des cartes étaient perdu au milieu du numéro des types définit au préalable pour les cartes. Pour y remédier, nous avons choisi de mettre le "printf" dans des macros que nous avons mis dans le fichier "cartes.h" qui prennent en paramètre la carte de correspondance et qui viendront remplacer dans le code juste avant la compilation.
Pour rendre ces fonctions d'affichages encore plus lisible, nous avons remplacé les entiers dans les "switch case" correspondant aux sous types des cartes par

des énumérations avec le nom de la carte, grâce à cela, en lisant le code, il sera plus facile de savoir qu'elle carte est traitée.

Après ces changements qui amélioraient juste la lisibilité du code, nous avons factorisé la fonction "add_distance()" qui traitait cas par cas l'ajout de distance jouée.

Il a été plutôt facile de factoriser cette fonction car en effet, chaque cas vérifiait la même condition principale qui était la vérification de possession de malus empêchant le joueur de jouer une carte distance. Seules les cartes distances 75, 100 et 200 avaient une condition en plus qui dans le cas où la vérification précédente n'était pas vrai, la carte jouée devait être inferieur ou égale à 50 si le joueur c une limitation de vitesse.

Puisque les cartes distances sont des multiples de 25, il nous suffit de multiplier 25 par la valeur représentant la carte distance jouée (sous types) pour obtenir la valeur réelle de la carte que le joueur souhaite jouer, sauf pour la carte 200 le résultat de cette multiplication sera de 125 et non pas de 200 carte il n'y a que 5 cartes distance et non 8. La carte 200 possède une vérification en plus qui lui est propre, l'impossibilité de poser plus de 2 cartes de cette valeur.

La dernière condition commune qu'ont les cartes distance est l'impossibilité de dépasser la distance nécessaire pour gagner: 1000.

Dans cette factorisation nous effectuons donc la premiéres vérification que toutes les cartes distances on en commun : la vérification de possession de malus, ensuite si le joueurs possède une limitation de vitesse nous vérifions que la carte distance jouée ne dépasse pas 50 (valeur de la limitation). Si la carte peut être jouée et ne dépasse pas la distance maximale (1000) alors nous incrémentons le compteur du joueur avec la valeur de la carte jouée. Dans le cas ou c'est une carte de 200 nous vérifions avant de la jouer si le joueur n'a pas déjà joué 2 cartes 200 au par avant.

Test

Pour finir, et s'assurer du bon fonctionnement de notre programme, nous avons réaliser une suite de test par macro dans notre code, qui permet de passer en mode "test" en modifiant la valeur de cette dernière. En effet, par le biais des fonctions se trouvant dans le fichier "test.c", nous allons tester si aucune des conditions seraient oubliées, ou bien bogguées, en les traitant cas par cas.

Le raisonnement étant identique, nous nous pencherons seulement sur le fonctionnement d'une de ces 4 fonctions: "test_malus_parade()"

Cette fonction permet de tester le bon fonctionnement des cartes attaques avec les cartes parades correspondantes. Pour cela, nous allons créer deux joueurs afin de simuler à la chaine toute les combinaisons malus/parade possible. Pour cela, on attribut au premier joueur le type de carte "Attaque", et au second le type de carte "Parade".
Enfin, à l'aide d'une double boucle, nous allons faire varier les sous types de cartes de chaque joueur afin de tester toutes les combinaisons possibles et inimaginables et nous allons faire jouer le joueur possèdant les cartes attaques grâce à la fonction "jouer_carte()".
Une fois cela fait, cette fonction retourne un entier. Si cet entier est négatif, c'est que nous avons à faire à une erreur.

Grâce à cela, il est maintenant possible de détecter facilement les coquilles dans nos conditions !

Conclusion

Pour conclure, ce projet nous a permis de prendre de bonnes pratiques méthodologiques en terme de code comme par exemple, d'utiliser des macros pour gérer nos différents cas d'erreurs, utiliser des énumérations pour définir nos types etc.
De plus, nous avons appris à mieux structurer notre code, de le rendre plus clair, et aussi d'apprendre à travailler en equipe.