Rapport sur le Projet FreeCell en C

Réaliser par : Abdenour Chenouf **encadrer par :** Mr. Abdelwahab Naji

Le projet FreeCell en C vise à implémenter le jeu de cartes FreeCell en utilisant des structures de données et des fonctionnalités du langage C. Le code est organisé en plusieurs fichiers pour faciliter la gestion et la compréhension du programme.

I. Structures de Données

Le projet utilise les structures suivantes :

a. Pile :

La structure Pile représente une pile de cartes. Chaque pile est utilisée pour stocker les cartes dans différentes zones du jeu.

b. Card :

La structure Card représente une carte avec un rang (A, 2-10, J, Q, K) et une enseigne (cœur, carreau, trèfle, pique). Les cartes sont liées entre elles pour former des piles.

c. Zone :

La structure Zone est utilisée pour représenter différentes zones du jeu, telles que le Deck, Temp, et Goal. Chaque zone est composée de plusieurs piles.

II. Initialisation du Jeu

Le jeu commence par l'initialisation du Deck, de Temp, et de Goal à l'aide des fonctions **initializePile** et **initializeZone**. Les cartes sont ensuite générées, mélangées et distribuées dans les piles du Deck à l'aide de la fonction **FillDeck** Cette fonction fait appel aux fonctions **generateFullDeck** qui génère 52 cartes 13 carreaux,13 piques, 13 trèfles et 13 cœurs puis les mélanger grâce au fonction **shuffleDeck**

J'ai déclaré un tableau d'enseigne (code UTF8) et un tableau de rang statiquement comme une variable Globale Pour affecter seulement les adresses aux suit et rang des cartes dans la fonction **CreateCard**

III. Affichage du Jeu

Les fonctions d'affichage telles que **Print_Deck**, **Print_Temp**, et **Print_Goal** permettent de visualiser l'état actuel du jeu. Les couleurs des cartes sont ajustées en fonction de leur enseigne, et les piles vides sont gérées correctement.

a. Print_Deck

Affiche l'état actuel des piles Temp.

b. Print_Temp :

Affiche l'état actuel des piles Temp.

c. Print_Goal :

Affiche l'état actuel des piles Goal.

Chacune de ses fonctions utilise La fonction **printPile**pour afficher toutes les cartes dans cette Pille(column) et cette dernière fait appel à la fonction **Print_Card**

Dans la fonction **Print_Card**j'ai appelé une autre fonction **SetCardColorsBySuit** qui change le fond de chaque carte en blanc et change la couleur d'écriture en noir ou blanc selon la 'suit' de chaque carte

La mise à jour de l'état de console est effectuée à l'aide de « **system("cls") »** qui supprime l'état de console pour laisser l'espace a la nouvelle état

IV. Mouvements de Cartes

Les mouvements de cartes sont gérés par plusieurs fonctions :

a. isMovetoTableValid :

Vérifie si le mouvement d'une carte d'une pile source vers une pile de destination dans le Deck est valide. Les règles du jeu, telles que les cartes devant être en ordre décroissant de rang et de couleurs opposées, sont respectées.

b. isMovetoGoalValid :

Vérifie si le mouvement d'une carte d'une pile source vers une pile de destination dans Goal est valide. Les cartes doivent être en ordre croissant de rang et de la même couleur.

c. moveCardInDeck :

Permet de déplacer une carte d'une colonne à une autre dans le Deck. Verifier la validité du mouvement Avant de l'effectuer.

d. **Move_Deck_To_Temp, move_Deck_To_Goal, Move_Temp_To_Goal, Move_Temp_To_Deck** :

Implémentent les mouvements de cartes spécifiques du Deck vers Temp, du Deck vers Goal, de Temp vers Goal et de Temp vers Deck respectivement. Vérifient la validité du mouvement avant de le réaliser.

Les mouvements sont effectuée grâce aux fonctions (**Pushcard,Pop et peek)**

V. Réinitialisation du Jeu

La fonction **resetGame** permet de réinitialiser le jeu en vidant les piles de Temp et Goal, puis en remplissant à nouveau le Deck.

VI. Conditions de Victoire ou Blockage

La fonction **checkwinner** vérifie si toutes les piles Goal contiennent exactement 13 cartes. Si c'est le cas, le joueur a gagné la partie.

La fonction `**checkBlocksituation**` évalue si la situation actuelle du jeu est bloquée en vérifiant plusieurs conditions :

1. Temp est-il plein ?

2. Mouvements possibles de Temp vers Deck ?

3. Mouvements possibles de Temp vers Goal ?

4. Mouvements possibles de Deck vers Goal ?

5. Mouvements possibles de Temp vers Goal ?

6. Mouvements possibles entre les colonnes de Deck ?

Si l'une de ces conditions est vraie, la fonction retourne `**false**` (pas bloquée). Sinon, elle retourne `**true**` (bloquée, aucun mouvement possible).

VII. Interface Utilisateur

Le programme propose une interface utilisateur simple en ligne de commande. L'utilisateur peut effectuer divers mouvements en choisissant parmi les options présentées à l'écran.

Cette jeu est 'user-friendly' grâce à l'utilisation de la fonction `**getValidChoice**`, qui assure une interaction conviviale en garantissant que l'utilisateur ne peut entrer que des choix valides, contribuant ainsi à une expérience de jeu plus fluide et intuitive.

VIII. Conclusion

En conclusion, ce projet de FreeCell en langage C démontre l'application efficace des structures de données pour créer un jeu de cartes interactif. L'utilisation de fonctions modulaires facilite la maintenance du code et offre une extensibilité pour d'éventuelles améliorations futures. La mise en œuvre des règles du FreeCell, combinée à une interface utilisateur conviviale grâce à des fonctions telles que `**getValidChoice**`, offre une expérience de jeu agréable. De plus, la logique derrière les fonctions de mouvement et les vérifications de situation assurent le bon déroulement du jeu tout en empêchant les mouvements invalides. En résumé, ce projet réussit à concilier complexité algorithmique, convivialité et respect des règles du jeu, fournissant ainsi une version numérique engageante du classique FreeCell.