Rapport projet CRAPS en C

Ce rapport présente de façon globale le fonctionnement et l'utilité de chaque fonction du projet ainsi que des deux bonus (implémentation d'une sauvegarde des joueurs et d'une sauvegarde des plus hauts gains) et apporte des éléments de précision sur certains éléments du code. Les fonctions sont présentées dans le même ordre que défini dans les fichiers « craps.c » et « craps.h » du projet.

Durant tout le programme on vérifie systématiquement la conformité des données entrées par l'utilisateur en lui demandant de rectifier son entrée si elle ne l'est pas. On considère en revanche que l'utilisateur n'a pas à modifier les fichiers de sauvegarde et si un fichier est corrompu le programme risque l'arrêt s'il est empêché de continuer (exit avec le code 1).

En vert sont indiqués les noms des fonctions présentes dans « craps.c » et dont le prototype se trouve dans « craps.h ». En orange sont indiqués les noms des variables et des pointeurs définis dans le programme. En rouge sont indiqués les noms des structures. Enfin en violet sont indiqués les noms des fonctions définies dans une des librairies standards incluses (à savoir stdio, stdlib et time).

Le programme a été conçu pour retourner différents codes de sortie en fonction de l'origine de son arrêt. La dernière page de ce rapport explicite les origines des codes de sorties retournés par le programme.

La compilation du code a été testé en particulier avec Visual Studio et Dev-Cpp sur l'OS « Windows 10™ » en respectant le standard ISO C99. Pour éviter tout bug (en particulier lié à la console Windows™) aucune lettre accentuée n'est affichée.

Note : La première ligne de « craps.h », (à savoir « #pragma warning(disable : 4996) ») permet de désactiver un avertissement bloquant levé par Visual Studio lors de l'utilisation de « scanf ». Elle est normalement ignorée par les autres compilateurs. À noter que l'on peut aussi supprimer cette ligne et utiliser « scanf_s » à la place de « scanf » mais le code ne devient alors plus portable d'un compilateur à l'autre (« scanf_s » étant une fonction spécifique au compilateur Microsoft™). Dans le code développé pour ce projet il n'y a de toute façon aucune différence de sécurité entre l'utilisation de « scanf » et « scanf_s » étant donné que nous ne récupérons qu'uniquement des entiers à l'aide de cette fonction et aucune chaine de caractère.

Structure joueur :

Structure permettant de stocker les informations relatives à un joueur : son nom, son nombre de jetons, si sa mise est placée sur « pass » (1) ou « don't pass » (0), le montant de sa mise et ses gains durant la partie.

Fonctionnement général d'une partie :

La partie débute sur le menu principal où l'on donne au joueur le choix entre commencer une partie, regarder les « highscores » ou quitter le programme. Lorsque l'utilisateur débute une partie il configure le nombre de joueurs puis chaque joueur est tour à tour invité à saisir son nom et le nombre de jetons dont il souhaite disposer pour cette partie. Pour modifier le nombre de caractères maximum pour le nom des joueurs on peut modifier la valeur de la constante « TAILLE_NOM » défini au début du fichier « craps.h » : si ce nombre est revu à la baisse par le programmeur il convient d'effectuer une suppression des éventuels fichiers de sauvegarde pour éviter un problème de compatibilité (aucun problème si le nombre est revu à la hausse). Selon la situation du jeu on peut demander au joueur de configurer sa mise et où il la place (« pass » ou « don't pass »). **À chaque fois que la phase 2 se termine** (= à chaque changement de lanceur) on propose de mettre fin à la partie ou de la sauvegarder. Si l'utilisateur met fin à la partie celle-ci se termine et les meilleurs scores sont sauvegardés s'ils entrent dans le top. La taille du top est configurable par le programmeur dans le fichier « craps.h » à tout moment en modifiant simplement la valeur de la constante « NOMBRE_GAINS ».

Si à la fin d'un tour le joueur choisi de sauvegarder sa partie celle-ci est **toujours considérée comme en cours** et les meilleurs scores ne seront donc enregistrés qu'au moment où cette partie prendra définitivement fin. Lorsque l'utilisateur voudra rejouer on lui demandera s'il souhaite commencer une nouvelle partie en mettant fin à celle sauvegardée (en enregistrant les meilleurs scores de la partie sauvegardée puisque celle-ci sera définitivement terminée) ou s'il souhaite charger la partie sauvegardée. Les données de sauvegarde d'une partie sont stockées dans le fichier « joueurs.txt » et les meilleurs scores sont stockés dans le fichier « gains.txt » dans l'ordre décroissant (les meilleurs scores au début).

Fonctionnement des différentes fonctions du programme :

Dés :

Simule simplement le résultat de 2 lancers de dés successifs en additionnant 2 nombres pseudo-aléatoires situés entre 1 et 6 obtenus grâce à la fonction rand.

Nombre :

Renvoie un entier qu'elle demande à l'utilisateur (ou qu'elle charge depuis le fichier de sauvegarde si celui-ci existe). Cet entier correspond au nombre de joueurs souhaité pour la partie à venir : il doit obligatoirement être égal ou supérieur à 1 (la fonction se répète jusqu'à ce que cette condition soit vérifiée). On utilise la fonction fseek dont on se sert pour « réinitialiser » le buffer clavier (en plaçant l'indicateur de position associé à l'entrée standard à la fin de ce qui est contenu dans le buffer) : cela permet en particulier d'ignorer le caractère de retour à la ligne généralement renvoyé par la console lorsque celui-ci n'est pas capturé comme ici avec scanf (et donc risque de récupération non souhaitée lors de l'utilisation d'un fgets ultérieur par exemple).

Comme dans tout le reste du programme nous nous assurons que les données entrées par l'utilisateur soient correctes. La procédure reste pour chaque cas qui se présente la même : on vérifie la conformité des données entrées par l'utilisateur et si ce n'est pas conforme on le signale à l'utilisateur, on vide le buffer clavier et on lui demande une nouvelle fois d'entrer des données jusqu'à ce qu'elles soient conformes à ce qui est attendu.

Dans le cas qui se présente ici on s'assure que la valeur retournée par scanf est bien 1 (= entrée correcte : l'utilisateur a bien entré un entier comme attendu) et si ce n'est pas le cas on vide le buffer clavier à l'aide de la fonction fseek et on redemande une entrée à l'utilisateur jusqu'à ce qu'elle soit correcte. On s'assure ensuite que le nombre de joueurs est bien supérieur ou égal à 1 sinon on lui demande une nouvelle fois.

Ajouter Joueur :

Si un fichier de sauvegarde existe on charge les informations depuis ce fichier.

Sinon demande à l'utilisateur un certain nombre d'informations (nom/jetons mis en jeu) pour pouvoir stocker ces informations dans la structure Joueur créée et renvoyée. En particulier on s'assure que l'utilisateur n'entre pas un nom qui dépasserait la taille autorisée grâce à la fonction fgets (taille que l'on peut paramétrer dans le ficher « craps.h » en changeant la valeur *TAILLE_NOM* de la directive correspondante). Si l'utilisateur entre un nom trop grand on ne récupère que les n premiers caractères (n correspondant à la valeur de *TAILLE_NOM*). Le caractère de retour à la ligne étant généralement capturé par fgets on s'assure de bien le supprimer (fin de la chaine de caractères dès que le caractère de retour à la ligne est rencontré). On vide ensuite le buffer clavier avec la fonction fseek (sinon les prochaines entrées clavier risquent d'être erronées si le nom entré est plus long que *TAILLE_NOM*). A noter que la fonction prend un paramètre permettant d'afficher le numéro du joueur dont on demande les informations.

Tableau :

Renvoie un pointeur pointant vers l'adresse d'un tableau **alloué dynamiquement** (en fonction du nombre de joueurs pris en paramètre) dont chaque « case » est une structure Joueur contenant les informations d'un joueur. L'ensemble du tableau contient donc les informations de chaque joueur. On alloue dynamiquement le tableau avec malloc (on s'assure une nouvelle fois que le nombre de joueurs soit supérieur ou égal à 1 sinon l'emplacement mémoire allouée avec malloc sera invalide). On vérifie par précaution que cette allocation s'est effectuée correctement sinon le programme s'arrête avec un code 3 (en particulier si l'OS nous refuse une allocation mémoire : on ne peut rien faire on quitte le programme). Enfin on remplit le tableau en récupérant pour chaque joueur la structure à stocker à l'aide de la fonction ajouterJoueur. **L'espace mémoire du tableau étant alloué dynamiquement on s'assure de bien libérer l'espace demandé lorsque le tableau n'est plus utile.** C'est pourquoi un free(tableauJoueurs) est bien effectué à la fin de la fonction main.

#BONUS 1 : Continuer :

Si un fichier de sauvegarde est détecté on demande à l'utilisateur s'il souhaite reprendre la partie sauvegardée ou commencer une nouvelle partie en mettant fin à celle sauvegardée. S'il met fin à la partie, la partie sauvegardée se termine : les résultats sont affichés, les éventuels plus hauts gains enregistrés et l'utilisateur est redirigé vers le menu principal. Dans tous les cas supprime le fichier de sauvegarde soit après avoir mis fin à la partie soit après l'avoir chargée si l'utilisateur la reprend. Le programme s'arrête (code 4) si la suppression est impossible.

Passe :

Enregistre pour chaque joueur s'il mise sur « pass » ou « don't pass » ainsi que la mise placée par le joueur (sauf celui pour lequel le dernier coup serait nul : ceux qui ont joué don't pass lorsque 12 est tombé *[Note : c'est pour cela que l'on prend un paramètre « nul » : pour savoir si le dernier coup était nul ou non]*). L'utilisateur doit entrer les lettres « P » ou « D » selon qu'il mise sur « pass » ou « don't pass ». On accepte aussi les chaines qui commencent par la lettre « P » ou « D » : on ne récupère que la première lettre de la chaine dans ce cas (donc si l'utilisateur tape « pass » ou « don't pass » cela fonctionne aussi).

Gagne :

Fonction assez simple qui est appelée lorsque le joueur gagne sa mise. Elle met à jour la structure Joueur dont l'adresse mémoire est prise en paramètre en ajoutant la mise du joueur à ses jetons et à ses gains.

Perd :

Même fonctionnement que la fonction ci-dessus mais on retire plutôt que d'ajouter. La fonction permet également au joueur de se définir un nouveau nombre de jetons quand celui-ci tombe à 0.

Phase1 :

Fonction qui gère la phase 1 du jeu : appelle la fonction des pour simuler deux lancers de dés, interroge les joueurs sur le placement de leur mise en appelant la fonction passe, puis en fonction de la valeur des lancers de dés effectue les actions prévues par le jeu pour chaque joueur. La phase 1 se répète jusqu'à ce que le point soit décidé (lorsqu'on a le point la fonction changementPhase est appelée).

ChangementPhase :

Appelle la fonction augmenter (si assez de jetons pour effectuer une augmentation) ou diminuer pour chaque joueur selon le placement de sa mise et appelle phase2 ensuite.

Augmenter :

Met à jour une structure Joueur dont l'adresse est passée en paramètre en augmentant la mise si le joueur le souhaite. On s'assure en particulier de vérifier si le joueur a bien les moyens d'augmenter sa mise du montant qu'il indique.

Diminuer :

Même fonctionnement que la fonction ci-dessus mais pour diminuer la mise. On s'assure en particulier de vérifier si le montant entré est positif et supérieur ou égal à la mise actuellement en place.

Doubler :

Demande à l'utilisateur d'entrer « O » pour doubler sa mise ou autre chose s'il ne souhaite pas la doubler. Met simplement à jour une structure Joueur dont l'adresse est passée en paramètre en doublant la mise ou en la laissant telle quelle selon le choix de l'utilisateur.

Diviser :

Exactement le même fonctionnement que ci-dessus mais pour réduire de moitié la mise.

Phase2 :

Comme Phase1 effectue différentes actions pour chaque joueur en fonction du résultat des lancers de dés. Prend en plus un paramètre « point » permettant de récupérer l'entier correspondant point (obtenu dans la phase 1) et un paramètre « suspension » permettant de savoir si la première suspension a déjà eu lieu (auquel cas la mise peut être réduite de moitié) ou non (la mise peut être doublée). Lorsque le tour est terminé (le point ou 7 a été obtenu) on propose de continuer la partie (une nouvelle phase1 se répète alors) ou de terminer la partie (la fonction choix est appelée).

Choix :

Propose à l'utilisateur de sauvegarder sa partie pour la reprendre ultérieurement ou de la terminer et appelle la fonction correspondant à son choix.

FinPartie :

Affiche les gains réalisés par chaque joueur, appelle la fonction sauvegardeGains (en mode 0 : voir le détail de la fonction sur la page suivante) qui sauvegardera les éventuels meilleurs gains réalisés durant la partie et redirige l'utilisateur vers le menu principal (« relancement » du programme en recommençant la fonction main).

#BONUS 1 : SauvegardeJoueurs :

On « ouvre » le fichier « joueurs.txt » en écriture avec suppression du contenu au préalable (si un autre fichier de sauvegarde est présent [ce qui n'est normalement jamais le cas sauf intervention de l'utilisateur car supprimé avec la fonction continuer plus tôt] les données présentes sont supprimées pour laisser place aux nouvelles). Dans le cas général le fichier « joueurs.txt » n'existant pas il sera donc créé. On enregistre ensuite toutes les informations nécessaires à un chargement de partie ultérieur dans le fichier : on sauvegarde sur chaque ligne les informations correspondant à un joueur soit ici respectivement son nombre de jetons, le gain réalisé jusque là et son nom. On indique à l'utilisateur si la sauvegarde s'est bien déroulée.

#BONUS 2 : TriGains :

Utilisée pour la réalisation de la fonction sauvegardeGains ci-dessous. On note qu'il n'est absolument pas gênant que cette fonction et celles qui suivent aient une complexité en temps ou en espace élevée car les tableaux manipulés sont très courts (seulement 15 éléments si l'on retient les 15 meilleurs scores par exemple). Cette fonction renvoie un tableau alloué dynamiquement (dont on a bien libéré l'espace mémoire après l'avoir utilisé dans la fonction sauvegardeGains) de la même taille que celui pris en paramètre contenant pour chaque élément sa position dans l'ordre croissant. Par exemple si le $3^{ème}$ élément du tableau passé en paramètre est le minimum alors le $3^{ème}$ élément du tableau que l'on crée dans cette fonction sera 0. Si le $4^{ème}$ élément du tableau passé en paramètre est le $6^{ème}$ plus petit alors le $4^{ème}$ élément du tableau que l'on crée dans cette fonction sera 5 (6-1 comme on commence à 0). Dans le tableau que l'on crée on veut de plus que chaque valeur soit unique car ces valeurs nous serviront à écrire les meilleurs scores dans l'ordre dans notre fichier « gains.txt » : on veut savoir quel score écrire en premier, puis en deuxième, etc. On s'assure donc en cas d'égalité entre 2 valeurs du tableau passé en paramètre qu'on ait 2 valeurs différentes successives (puisqu'égalité) dans notre nouveau tableau : on notera que l'ordre en cas d'égalité n'a aucune importance : par exemple si on a 3 minimums dans le tableau passé en paramètre on peut leur attribuer 0, 1 ou 2 dans le nouveau tableau peu importe qui a 0, 1 ou 2 : en revanche il faut faire en sorte que 2 éléments ne soient pas attribués au même chiffre.

#BONUS 2 : IndiceMin :

Teste simplement toutes les valeurs du tableau passé en paramètre (sans risque de débordement puisque lorsque la fonction indiceMin est lancée on sait que la taille du tableau est égale à « NOMBRE_GAINS ») et renvoie l'indice de l'élément minimum du tableau passé en paramètre.

#BONUS 2 : SauvegardeGains:

Cette fonction peut avoir plusieurs rôles selon la valeur de l'argument mode pris en paramètre. Le fichier « gains.txt » contient sur la première ligne le nom de la personne ayant réalisé le score le plus haut puis son score sur la deuxième ligne, sur la troisième ligne le nom de la personne ayant réalisé le deuxième plus gros score et son score sur la quatrième ligne et ainsi de suite.

Si mode==0 : la fonction va enregistrer les highscores du tableau de joueur pris en paramètre

Si mode==1 : la fonction va simplement lire le fichier « gains.txt » pour afficher les highscores. Dans ce mode si le fichier n'existe pas un message indiquera qu'aucun score n'a jamais été enregistré.

Dans les deux modes, si un fichier existe il va être lu et les éléments contenus dans celui-ci vont être stockés dans deux tableaux : un tableau contenant les noms des personnes présentes dans le fichier et un autre contenant leurs scores (= gains réalisés en fin de partie). Les deux tableaux sont conçus de telle sorte que chaque indice corresponde à un des highscore enregistré (c'est-à-dire que par exemple tableauNom[2] correspond au nom de la personne ayant réalisé le score contenu dans tableauGains[2]). On en profite également pour stocker le nombre de highscores lu dans le fichier dans la variable nombreEnregistrement (0 si le fichier n'existait pas).

Si mode==1 on affiche simplement le contenu des deux tableaux et la fonction se termine.

Sinon si mode==0 on va enregistrer les éventuels scores présents dans le tableau pris en paramètre qui se qualifierait pour rentrer dans les highscores. Etant donné qu'il faut d'abord savoir pour chaque score s'il convient de l'ajouter au fichier ou non (selon qu'il fasse partie des meilleurs scores ou non) on va d'abord modifier nos tableaux présents localement dans notre fonction avant d'écrire dans le fichier.

Si le fichier « gains.txt » n'est pas saturé (on entend par là que le nombre de gains que l'on a décidé de mémoriser (NOMBRE_GAINS) est supérieur au nombre de gains présents dans le fichier) il nous suffit d'ajouter le score à la fin puisqu'il fait de toute façon partie des meilleurs scores. En revanche si le fichier est saturé il faut connaître le score minimum présent dans celui-ci pour savoir si notre score déclasse celui-ci ou non. On utilise alors la fonction indiceMin qui nous permet de trouver l'indice de notre tableau qui contient l'élément minimum et on compare notre score à ce score minimal : s'il est plus grand on le remplace sinon on laisse tomber et on passe au score suivant.

Une fois cela effectué on aimerait enregistrer nos scores du plus grand au plus petit. On récupère alors l'adresse d'un tableau tri obtenu grâce à la fonction triGains (voir détails sur la page précédente) ce qui nous permet d'écrire dans le fichier les scores du plus grand au plus petit (et donc de les afficher dans cet ordre lors de l'affichage).

Code de sortie :

Code de sortie 0 : Le programme s'est terminé normalement

Code de sortie 1 : Un fichier de sauvegarde de la partie est manifestement corrompu et empêche la continuation du programme

Code de sortie 2 : Une allocation mémoire ne s'est pas déroulée correctement (non attribution de l'emplacement demandé par l'OS) empêchant la bonne continuation du programme

Code de sortie 3 : La valeur d'une variable a été altérée empêchant la bonne continuation du programme

Code de sortie 4 : Erreur lors de la suppression du fichier de sauvegarde de la partie (vérifier si l'exécutable ne s'exécute pas depuis un dossier sécurisé en écriture).