

Année scolaire : 2020-2021

**Rapport de projet**

Une image contenant texte

Description générée automatiquement



Professeur référent : M.George

Pre-ing1 Groupe 6B: PAUL JUMEL, YASMIN HAJI-MOHAMMED, KEVIN RAHMANI

**Sommaire**

* I. INTRODUCTION
* II. RESPECT DU CAHIER DES CHARGES
* III. REPARTTION DES TACHES ET ORGANISATION
* IV. AVANTAGES ET LIMITES/DIFFICULTES
* V. CONCLUSION

# Introduction

**Choix du jeu :**

* Pour le projet d’informatique des élèves de Pré-ING 1, nous avons communément décidé de choisir le « *6 qui prend* », un jeu de carte qui a marqué notre enfance. Joué par les enfants comme par les grands les règles du jeu sont très simples à comprendre ce qui le rend accessible à tous. Sans pour autant perdre le côté compétitif, car de nombreuses stratégies existent afin de prendre l’ascendant sur ses adversaires et de remporter la partie.
* Voici un cours résumé du jeu : En début de manche chaque joueur reçoit 10 cartes. À chaque tour, les joueurs choisissent une carte et la révèlent en même temps : ces cartes sont ajoutées à l’une des 4 séries qui se forment sur la table. Celui qui doit jouer la sixième carte d’une série « récolte » les 5 premières... et toutes leurs têtes de bœufs (respectivement attribuées à chaque carte) ! Quand les 10 cartes sont jouées, chacun compte ses bœufs et les additionne à son total précédent. Après plusieurs manches, le plus petit troupeau gagnera la partie.

**Respect du cahier des charges**

* **Objectif principal : implémenter toutes les règles**

Le programme a été réalisé en langage C à l’aide de Code Blocks, avec les bibliothèques : <stdio.h>, <stdlib.h> qui nous permettent d’utiliser les fonctions de bases, <time.h> pour créer des nombres aléatoires et enfin <windows.h> pour les « *sleep* ».

Le « 6 qui prend » comprend de nombreuses règles, nous avons donc dû créer beaucoup de fonction afin que ces dernières soient respectées. En voici une liste exhaustive, accompagnée d’information, arrangée pour correspondre au déroulement du jeu :

**Début de partie : Création et distribution des cartes et du board.**

**Fonction :**

**Int nbRandom** : fonction prédéfinie vue en cours qui permet de choisir un nombre aléatoire entre deux extrémums. Ici, c’est 1 et 104

**Void iniTabJoueur :** fonction qui crée le deck du joueur, utilisation d’un tableau dynamique à double entrée dont les lignes servent à stocker les cartes et les colonnes le numéro du joueur. Création de 10 cases vides initialisées à 0. Afin de se déplacer dans celui-ci on utilise une double boucle for avec i, j variable locales.

**Void iniTabDeck :** fonction qui permet de stocker à chaque tour les cartes choisis par les joueurs. Utilisation d’un tableau dynamique et d’une simple boucle for qui doit s’arrêter jusqu’à ce que i soit inférieur au nombre de joueur avec incrémentation à chaque tour (pour des raisons de commodité les détails des boucles for ne seront plus précisés)

**Void afficherTableau :** fonction qui affiche un tableau à double entrée

**Void afficherBoard :** fonctionqui analyse les cartes présente sur le plateau du jeu. A l’aide d’une double boucle for, nous pouvons nous déplacer dans le board. Ceci nous permet qu’à chaque fois qu’une nouvelle carte est ajouté lorsque les joueurs misent ou au début, de pouvoir afficher la valeur de la carte présente sur le board à l’aide de condition if et d’un printf. Utilisation de modulo pour trier les cartes divisibles par 10 et 5 qui valent 3 pts, modulo 5 seulement = 2 pts, les cartes comme 11, 22, 33 etc. qui valent 5 pts, 55 qui vaut 7 pts, et toutes les autres ne remplissant pas ces conditions sont initialisées à 1 pt.

Ainsi les joueurs peuvent voir combien de points une ligne contient s’ils doivent la choisir. De plus, nous avons attribué à chaque ligne un numéro afin que le board soit plus lisible. Pour la mise en forme nous avons séparé chaque « case » par des | pour qu’il y ait une démarcation entre ces dernières (ajout de \n également pour faire un peu d’espace).

**Void tri\_selection :** fonction de tri lent vu en cours pour un tableau dynamique à deux dimensions

**Void tri\_Deck :** fonction de tri lent pour un tableau dynamique à une dimension

**Int test :** fonction qui teste si les cartes du deck ne présente aucun doublon

**Int testBoard :** fonction qui teste si les cartes de début de jeu déposé sur le board ne présente pas de doublon et sont uniques

**Void distribution :** fonction qui distribue les cartes aux joueurs à l’aide d’une double boucle for et d’une boucle while. On utilise la fonction nbRandom pour générer les cartes et la fonction test pour s’assurer de l’unicité. De plus on utilise la fonction tri selec pour ranger le deck du joueur.

**Void initboard :** fonction qui initialise le board, tableau à double dimension de 4 lignes et 5 colonnes dont toutes les valeurs de bases sont initialisées à 0, comme la fonction distribution on utilise la fonction nbrandom pour générer des nombres aléatoires pour la première colonne et la fonction testBoard pour s’assurer que les cartes soient bien jouables (utilisation d’une double boucle for et d’une boucle while pour s’assurer que la première colonne du tableau soit remplie).

**Milieu de Partie :** mise en jeu des cartes

**Void afficherDeck :** fonction qui affiche le deck du joueur, utilisation d’une double boucle for et d’un printf

**Int testValide :** fonction qui teste si la carte joué par le joueur est bien présente dans son deck (boucle while pour marquer une condition d’arrêt avec double boucle for pour tester tout le deck et utilisation d’une boucle if condition de réussite pour que la carte soit bonne).

**Void remplacer :** fonction qui remplace la carte joué par le joueur lors de chaque tour. Lorsque cette dernière est jouée, la case qu’elle occupe dans le deck du joueur est remplacé par la valeur 0 et la valeur de la carte également (double boucle for et boucle if).

**Void duplique :** fonction qui crée un second board mais non visible par les joueurs reprenant les mêmes dimensions que l’original. Cependant, ce dernier ne reprend que la dernière carte présente sur sa ligne et ignore les autres. Son utilité réside dans le fait qu’avec lui, nous pouvons comparer bien plus facilement les cartes jouées par les joueurs et les cartes présentes sur le board pour savoir où ces dernières vont aller (dans quel ligne et colonne). Pour cela, nous avons repris la même structure algorithmique que initboard en rajoutant des conditions if pour repérer les dernières cartes et les attribuer aux lignes et colonnes correspondantes du second board.

**Void choisirCarte :** une des fonctions primordiales du programme qui permet de déposer la carte joué par un joueur sur le board. On demande d’abord au joueur de confirmer sa présence en appuyant sur 1 à l’aide d’un printf/scanf. Si ce dernier répond présent le système lui affiche son deck en utilisant la fonction afficher deck, affiche son nombre de points en utilisant tabtransforme et enfin affiche le board en utilisant la fonction afficher Board. Une fois cela fait le système demande au joueur de choisir une carte à jouer à l’aide d’un printf/scanf et vérifie cette dernière avec la fonction test Valide. Si cette dernière peut être joué alors la fonction remplacer prend le relais et supprime la carte joué du deck du joueur. La même opération est effectuée pour les cartes joués par l’IA sauf qu’on ne lui demande pas de choisir ses cartes. Une fois les cartes jouées, elles sont stockées dans tabdeck et rangés dans l’ordre croissant avec tri deck. Un second tabdeck 2 est créé pour stocker les cartes des joueurs mais on ne les tries pas

**Int stock :** fonction qui permet de retrouver grâce à Tabdeck2, à qui appartient la plus petite carte jouée.

**Int stock2**: fonction qui permet de retrouver à qui appartient la carte jouée

**Void transforme :** fonction qui compte le nombre de points sur la ligne prise et qui additionne ces points aux nombres de points que le joueur possède. On utilise une structure similaire à afficherBoard pour le comptage de points et utilisation de stock pour retrouver le joueur qui a joué la + petite carte

**Void transforme2 :** fonction qui compte le nombre de points sur la ligne prise et qui additionne ces points aux nombres de points que le joueur possède. Même structure que la précédente mais on utilise stock2 cette fois-ci

**Void point\_ligne**: fonction qui calcule le nombre de points sur chaque ligne afin d’aider l’IA à choisir la ligne qui possède le moins de points. Sa structure algorithmique est semblable à celle de afficherBoard.

**Void remplirBoard :** une des fonctions primordiales du programme qui sert à remplir le board. Dans un premier temps, création d’un tableau(tabBoard) qui permet de stocker les valeurs inférieures à la carte joué par le joueur. Nous attribuons à a la valeur 1, puis nous trions tabBoard dans l’ordre croissant. Ensuite on récupère la dernière valeur du tableau puis on l’enregistre dans 2 variables ses positions (numéro ligne et colonne)

Dans un second temps si la première carte (la plus faible) n’est supérieure à aucune carte présente sur le board (se traduit par a=0).On regarde alors à qui appartient cette carte et le joueur/IA en question choisi une ligne. On remplace ensuite cette ligne du Board par des 0 sauf à la première colonne où on place la carte la + faible joué, appel de transforme pour compter le nombre de points que se prend le joueur/IA. Si la valeur de la colonne est égale à 4 (dernière colonne) on fait la même chose qu’avant

Pour finir si on rentre dans aucune de ces conditions on ajoute la carte dans le Board en mode normal.

**FIN DE PARTIE :** condition d’arrêt

**Int fin :** fonction qui marque la première condition d’arrêt du programme, c’est-à-dire une des conditions pour lesquelles le joueur en question perd et ne peux plus joueur, en l’occurrence si ce dernier atteint 66 têtes de bœufs. Utilisation d’une double boucle for qui vérifie cette condition pour chaque joueur.

**Int fin2 :** fonction qui permet de renvoyer le numéro du joueur possédant le moins de point.

**IA :**

**Int choisirCarte\_IA :** fonction qui permet de choisir automatiquement une carte du deck de l’IA. Afin de choisir la carte, on la compare aux cartes du second board de duplique puis on essaie de trouver une carte supérieure aux cartes présentes sur le Board. Utilisation d’une triple boucle for et d’une condition if qui si on ne trouve pas de carte supérieure on prend celle qui est la plus petite.

**MAIN :** explication

-Déclaration des variables min et max du type int correspondant à 1 et 104

-Déclaration d’autres variables nécessaires au programme dont : nbJoueur, IA, choix, ContinuePartie et back

On crée les tableaux dynamique de tabCompt ( compte le nombre de points sur les lignes du board, sert pour l’IA), tabdeck (tableau qui garde les cartes jouées à chaque tour trié), tabdeck2 (pareil mais non trié), tabBoard ( repère sur quel ligne et colonne la carte joué doit être placée). On crée ensuite le board, tableau à cinq colonnes et quatres lignes et on initialise ses valeurs à 0. On crée également le second board de la fonction duplique. Ensuite on crée 3 autres tableaux :

-Création du tableau TabTransforme pour le nombre de points des joueurs

-Création de Tabjoueur, deck des joueurs

-Création de TabStock stocker les valeurs de la ligne prise

On entre, après cela, dans la seconde partie du main, établi à l’aide d’une boucle while.

A l’aide de printf nous faisons tout d’abord un peu de mise en forme pour donner au menu une belle présentation. Nous demandons ensuite avec un scanf de rentrer un numéro entre 1 et 3 en fonction de ce qu’il veut faire : 1 lance le jeu, 2 pour les règles du jeu et 3 pour Quitter.

Ces choix sont desservis aux joueurs grâce à un switch que nous allons examiner de plus près .

**Premier cas :** Le premier cas lance donc une partie, on demande à l’hôte combien de joueurs et d’IA veut-il pour cette même partie. On appelle ensuite la fonction IniTabJoueur pour l’appliquer aux tableaux crée au-dessus. On enchaîne avec l’appel de distribution pour les cartes, et initboard pour le board (afin de le crée). On crée par la suite la variable k qui va nous servir de compteur de tour. Utilisation d’une boucle for. On appelle les 2 fonctions qui permettent le bon déroulement du jeu c’est à dire ChoisirCarte et remplirBoard. Puis on appelle fin pour vérifier à chaque tour si un joueur n’a pas plus de 66 points. Si au bout de 10 tour (k=10), personne n’a plus de 66 points c’est celui qui en a le moins qui remporte la partie

On demande enfin à l’hôte si ce dernier veut relancer directement une partie ou non à l’aide d’un printf/scanf qui relance la boucle do while si on rentre 1

**Deuxième cas :** On explique les règles du jeu, tout simplement.

**Troisième cas :** On sort du programme avec back=0

**Défaut :** on printf à l’utilisateur de réessayer car son chiffre est invalide

Répartition des tâches et organisation

* Lors du lancement du projet en février/mars, la situation dans laquelle nous étions en raison de la crise sanitaire nous a grandement handicapés sur l’avancement du programme car des sessions de travail ensemble étaient devenues impossible à organiser (couvre-feu). Cela a grandement impacté notre motivation mais nous savions d’ores et déjà qu’il fallait absolument travailler car ce projet contient deux notes d’une matière importante qui pourrait consolider notre passage en 2ème année.
* Nous avons donc travaillé de manière régulière afin de ne pas être submergé par la masse de travail dû à un potentiel retard. Le travail fourni était sérieux et à chaque session virtuelle nous avancions de manière considérable. Paul a énormément travaillé sur les fonctions et sur la fonction main à l’aide de Kevin. Tandis que Yasmin avec le soutien des deux autres membres s’est consacrée au rapport. La répartition du travail était homogène, si quelqu’un travaillait plus sur quelque chose quelqu’un d’autre rattrapait le lendemain pour équilibrer ce qui a permis une bonne atmosphère de travail tout au long de cette période. Pour parler concrètement, les heures de travails avoisinaient les 4h par semaine. Cependant nous les avons doublés pendant les vacances grâce au temps disponible, mais réduites lors des semaines chargées (cf. DS algèbre & analyse). Notre planning de travail se résumait aux heures de cours le mardi et mercredi et parfois les week-ends, que nous avons totalement respecté.

**Avantages et limites / difficultés**

**Avantages :**

* De manière totalement objective, notre programme ne contient que des avantages.   
  En effet, pour le plaisir de vos yeux, nous avons créé un menu regroupant trois options :
  + 1. Joueur au jeu,
  + 2. Lire les règles du jeu,
  + 3. Quitter.
* Les nouveaux joueurs peuvent donc prendre le temps de bien lire et comprendre les règles du jeu avant d’y jouer. Une fois arrivé dans la partie, nous avons établi une belle présentation du « *deck*» de chaque joueur afin que ces derniers voient clairs dans leurs jeux. Le « *board* » est également bien présenté et lisible, nous avons mis pour chaque joueur un compteur de tête de bœufs qui leur permettra de voir ou ils en sont. Un « *sleep* » (à retirer) a également été ajouté entre deux joueurs ainsi que la nécessité de confirmer la présence du joueur en question en appuyant sur « 1 » au début de chaque tour pour éviter toute potentielle triche.
* Si le joueur veut refaire une partie à la fin, nous avons ajouté la possibilité d’en relancer directement une sans repasser par le menu pour continuer à faire vivre la compétitivité présente. Si le joueur n’a pas d’amis, nous avons pensé à tout : il pourra jouer avec une ou plusieurs IA et s’amuser à gagner contre elles. Vive la solidarité !

**Limites :**

* La seule limite présente dans notre programme est le niveau des IA. En effet, elles sont réglées pour jouer les actions les plus prévisibles. Ainsi même s’il ne s’agit qu’un jeu de hasard. Un joueur expérimenté battra facilement l’IA.

**Difficultés**

* Les fonctions qui nous ont été particulièrement difficile à coder sont : la fonction choisirCarte, principalement pour sa rédaction et non pour son sens mais l’inverse a eu lieu pour la fonction remplirBoard en raison de sa longueur et de son importance primordial au programme. D’autres problèmes mineurs ont été rencontrées dans les fonctions transforme et transforme2 mais nous avons pu les résoudre grâce à M.George.
* Les plus grosses difficultés ont été rencontrées au moment de la création de l’IA car il fallait revoir tout notre programme et changer les conditions d’arrêts des doubles boucle for qui étaient supposés s’arrêter en fonction du nombre de joueurs. Il fallait donc qu’elle s’arrête aux fonctions du nombre de joueurs IA comprise(s), ce qui était très long et fastidieux à faire vu que le programme faisait 600 lignes à ce moment-là. De plus, de nombreux bug de compilation sont apparus pour X raisons lors du lancement du programme avec l’IA, le menu ne voulait plus s’afficher, nous avons donc du tout revoir et tout corriger et espérer que ça fonctionne.

**Conclusion**

* En conclusion, ce projet nous a permis de considérablement augmenter notre niveau global en codage et notre autonomie car nous avons dû nous-mêmes réfléchir à la transcription algorithmique des règles du jeu, leurs ordres, leurs efficacités et tout un tas d’autres facteurs.

Cela a été une belle expérience et un petit avant-goût de ce qui nous attend pour certains d’entre nous. Au-delà du projet, nous pensons que cela a permis de se conforter sur nos choix de continuer ou non dans le codage. Le projet a également été bénéfique pour notre future orientation.

* Un grand merci à M.George, au groupe et à Cy Tech

Paul, Kevin, Yasmin.

Pré-ING 1 groupe 6b.

2020-2021.