

Eliass Hajajou
Yassin M'Chaar
Amine Hajir
6 janvier 2015

Rapport de projet :

Puissance 4 ++



Sommaire :

3	Introduction au projet.....
	a) Recherche
	b) Organisation des idées
	c) Prévisualisation
6	Conception.....
	a) Étude du puissance 4++
	b) Découpage du problème en fonctions
	c) Écriture
10	Mise en place du SDL.....
13	Tests et problèmes.....
	a) Les problèmes
	b) Les solutions
16	Conclusion.....

I / Introduction au projet

Avant de commencer les recherches sur le projet il nous a fallu choisir un sujet dans une liste donnée.

Parmi tous les sujets , « puissance 4 ++ » nous semblait le plus intéressant, nous l'avons donc placé en première place dans nos choix.

Lors de la remise des sujets nous avons eu le plaisir de pouvoir travailler sur ce sujet !

a) Recherche sur le projet

Nous connaissons tous le puissance 4 classique, mais nous ne connaissons pas le puissance 4++.

Il nous a donc fallu faire quelques recherches sur les règles de jeu de base, puis créer nos propres règles.

Comme dans un puissance 4 classique la grille garde une taille 6x7 (6 lignes et 7 colonnes), on a donc gardé la même taille de grille pour faciliter la mise en place du plateau.

Nous avons choisi d'imposer un nombre limité de joueur, on aura donc le choix de jouer de 2 à 4 joueurs.

Aussi, on avait à disposition trois types différentes de pièces : les pleines, les creuses et les bloquantes:

Les pleines : ces pièces peuvent se superposer sur une pièce creuse (mais pas bloquante ni pleine).

Les creuses : ces pièces peuvent se superposer sur une pièce pleine (mais pas bloquante ni creuse).

Les bloquantes : empêchent de pouvoir jouer une pièce pleine ou creuse en dessous de cette pièce et au même niveau.

Aussi, un système d'aide a été mis en place,
Le jeu est gagné lorsqu'il y a 4 pièces alignées horizontalement, verticalement ou en diagonale. Sinon la partie est nulle (puisque toute la grille sera remplie sans gagnant).

b) Organisation des idées

Pour mener à bien notre projet, nous avons tout d'abord, sur papier écrit nos idées, qui permettront de répondre aux exigences des futurs utilisateurs de notre projet.

La partie la plus importante ici était d'avoir une idée précise, et concrète de notre structure concernant les pions, leurs symbolisations, leur affichage, le nombre de joueur, taille de la grille...

Pour cela, il a fallu non pas seulement être dans la peau du programmeur mais aussi des utilisateurs, car comme nous l'avons appris lors des cours de « Conduite de projet », ce programme doit répondre à des attentes précises.

Ici le programme devait permettre de jouer à plusieurs(jusqu'à 4 joueurs), mais nous devions aussi pouvoir gérer une liste des scores avec le nom des joueurs, un enregistrement d'état de la partie **et une demande d'aide afin de laisser le programmer chercher une solution à la place de l'utilisateur.**

Nous avons ensuite, regroupées les idées similaires et former deux groupes.

Par exemple : menu et l'affichage sera dans un même groupe.

Pour chaque groupe nous avons évalué la difficulté de conception, c'est à dire que nous devons mettre une note (1,3,5,8) afin que chaque membre du binôme puisse donner son point de vue sur la complexité du groupe.

Une fois tout ceci établi, nous nous sommes réparties les tâches, une personne s'est occupée du déroulement du programme de base, c'est à dire la fonction jouer, l'initialisation de la grille, l'affichage de la grille, et l'autre personne de toutes les petites fonctions qui seront amenées à être utiliser dans le programme pour simplifier le code et la fonction victoire. Ensuite nous avons ensemble travailler sur la conception du menu, l'affichage, et le déroulement du jeu.

II / Conception

a) Étude du puissance 4++

Le puissance 4++ se joue sur un plateau de jeu 6x7. Nous avons donc du réfléchir (très rapidement) à la manière dont nous allions matérialiser ce plateau de jeu afin de pouvoir ensuite travailler dessus (placer des pions, puis tester les colonnes, les lignes et les diagonales...).

L'idée la plus simple au départ pour nous était de travailler sur des matrices d'entiers.

En effet elles sont simples à manipuler.

Mais lors des séances, du fait de l'utilisation de plusieurs type de pions, nous avons remarquer qu'il était préférable d'utiliser des matrices de caractères.

Il nous faut donc une matrice de taille 6x7 qu'on initialisera avec des « VIDE » qui seront ensuite défini par des « espaces », défini comme ceci dans notre programme « ' », soit « #define VIDE ' ' » .

De plus nous savons que ce puissance 4 ++ se joue avec 3 types de pièces différentes, donc chaque case de la matrice doit comporter un emplacement « pleine », « creuse », et « bloquante ». La solution qui nous est venue à l'esprit immédiatement est d'utiliser des structures dans cette matrice.

Lors de l'initialisation de la matrice il faudra donc initialiser les 3 champs de chaque case de la matrice !

On aura donc une grille de 7 colonnes, dont chacune de ces colonnes sera décomposée dans le programme en 3 sous-colonnes, une colonne « creuse », « pleine » et « bloquante ».

De là nous avons maintenant le plateau de jeu de base du puissance 4++ .

b) Découpage du problème en fonctions

Maintenant que nous avons résolu le problème de la modélisation du plateau de jeu (à travers une matrice de structures), nous n'avons plus qu'à réfléchir aux futures fonctions que nous allons écrire.

Nous avons convenu de nous occuper de toute la gestion du score, enregistrement de l'état de la partie, et de l'aide dès que le puissance 4++ serait opérationnel c'est-à-dire lorsqu'on pourrait jouer une partie, avec détection de victoire horizontale, verticale, diagonale ainsi qu'un match nul avec une saisie et un affichage au plus proche de notre but.

Nous avons donc décidé de nous concentrer principalement sur le placement des pions, les tests de victoire, colonne pleine, coups possibles, coordonnées valides.

Nous avons auparavant regrouper nos idées écrites, il nous restait plus qu'à les traduire en fonctions.

Il nous faudrait 3 groupes de fonctions : le premier groupe qui va être utilisées pour manipuler la matrice, le deuxième groupe qui va permettre la gestion des joueurs avec l'enregistrement de la partie dans un fichier et le troisième groupe contiendra le programme principale qui permettra de jouer.

Dans le premier groupe nous pensions donc mettre :

- Affichage titre du jeu et menu
- l'initialisation de la matrice
- affichage de la matrice titre du jeu et menu
- placement des pions
- test lignes, colonnes et diagonales
- test de la présence d'un pion à un endroit précis
- test grille et colonne pleine

Les fonctions initialisation de la matrice(void initialiser_jeu) et affichage(void afficher_jeu, et void afficher_colonne) permettent d'initialiser la matrice avec des « ' », espaces vides, afin de pouvoir les utiliser par la suite, et d'afficher cette matrice à l'écran.

A chaque tour de jeu, un joueur va poser un pion, pour cela il lui faut une fonction pour placer ce pion ce pion tombera au fond du plateau. Nous sommes partis dans un premier temps sur une seule et même fonction « jouer », qui prenait en paramètre le type de pion tapé par l'utilisateur, et placer le pion dans notre matrice.

Finalement, nous avons opter pour créer 3 fonctions booléennes, qui chacune sera destinée aux différents type de pion.

Ces fonctions qui permet de tester si 4 pions d'un joueur sont alignés en ligne, colonne ou en diagonale a prit beaucoup plus de temps.

Nous avons d'abord commencé par séparer ce problème en une seule et unique fonction qui teste chaque case de la grille et qui regarde si elle est suivie de 3 autres pions de la même couleur

Les 3 premières fonctions n'ont pas posé de problème lors de l'écriture cependant les deux dernières fonctions a été plus complexe.

Après plusieurs heures passées dessus, nous avons remarqué qu'il était préférable de décomposé en 3 fonctions booléennes (booleen placerPleine, booleen placerCreuse et booleen placerBlocante).

Chacune de ces fonctions prends en paramètre la grille du jeu, un paramètre qui vérifie si la colonne est pleine, un paramètre qui est saisi par l'utilisateur qui indique le numéro de la colonne à jouer, le numéro du joueur courant qui joue, le nom du joueur et enfin un paramètre qui renvoie vrai si le joueur a bien placé le pion sinon faux.

La fonction "grille_pleine" permet de vérifier que la grille est pleine et ainsi déclarée une partie nulle. Nous avons aussi du limiter le nombre de pions dans une colonne et donc créer une fonction « int hauteurColonne[M] » qui permet de savoir si une colonne est pleine et d'indiquer au joueur de jouer dans une autre colonne.

Cette fonction créer un tableau qui permet de stocker dans chacune de ses cases l'indice de la première case vide de la colonne correspondante dans le cas où la colonne est pleine, la valeur de la case correspondante à cette colonne dans le tableau est égale à 0.

Au début du jeu toutes les cases du tableau sont initialisées à N-1(5) car l'indice de la première case vide de chaque colonne est égale à l'indice de la dernière ligne de la grille.

Pour faire jouer tour à tour les joueurs, nous avons utiliser directement dans le main une variable de type int « player » qui permet de passer au joueur suivant en utilisant le système d'une boucle for.

Dans le deuxième groupe se trouverait :

- créer fichier
- créer joueur
- mise à jour du score des joueurs
- afficher le score
- menu
- lire les règles du jeu qui se trouve dans un fichier extérieur appelé « regles.txt »

Dans le troisième groupe se trouve le menu du jeu !

i) Écriture

Afin d'avoir un programme plus lisible (et plus léger en ligne) et qu'on puisse réutiliser ultérieurement, il nous a été conseillé de créer plusieurs fichiers (.c et .h header). De plus lorsqu'on devait modifier une fonction qui était utilisée dans plusieurs autres fichiers il nous suffisait de la modifier à un endroit (au lieu de chaque endroit où elle était utilisée). Nous avons donc commencé par créer un fichier « fonction.c », « fonction.h » et « main.c » qui contiennent respectivement toutes les fonctions de base de la matrice et tout le programme principal nécessaire pour jouer une partie.

Dans le main.c nous avons fait un « #include "fonction.h" » afin de pouvoir utiliser toutes les fonctions du fichier « fonction.c » (notons que le fichier « fonction.h » contient uniquement le nom de chaque fonction ainsi que les variables globales. Exemple : void initialiser_grille(Pion grille[N][M]);).

Pour pouvoir compiler tous nos fichiers, nous avons utilisé un makefile en y ajoutant des options telles que "-Wall" afin de mettre en évidence les avertissements et ainsi pouvoir les corriger et avoir un programme sans erreur d'exécution ni avertissements.

Lorsque nous avons commencé à écrire le programme nous nous sommes d'abord occupés de toute la gestion du plateau de jeu.

Comme nous l'avons dit précédemment nous avons créé une matrice de structures. Chaque structure contient 3 champs que l'on a appelé 'char pleine ; char creuse ; char bloquante'. Pour faciliter toute l'écriture du programme dans un premier temps nous avons initialisé chaque champ de la matrice avec un ' ', soit un caractère, vide.

Une fois les fonctions 'initialiser_jeu' et 'initialiser_hauteurColonne' terminées nous avons créé les fonctions 'afficher_jeu' et 'afficher_numColonne_jeu' qui affichera le plateau de jeu et le numéro des colonnes à chaque tour de jeu. Ces fonctions se sont écrites très rapidement.

Une fois le plateau affiché à l'écran il fallait pouvoir jouer un pion. Pour cela nous avons créé 3 fonctions, pour nos 3 types de pion 'placerPleine', 'placerCreuse' et 'placerBloquante'.

Pour ces fonctions il nous a fallu nous rappeler que le pion joué tombait tout en bas de la grille, et par notre fonction créer au préalable 'initialiser_hauteurColonne', on reçoit en paramètre l'indice de la hauteur de la colonne.

Cependant on ne peut plus jouer dans une colonne si celle-ci est pleine, nous avons donc grâce à la fonction précédente, indiqué au joueur un message en rouge et en gras pour alerter qu'il ne peut soit jouer qu'une pleine (si la dernière case de la colonne contient une creuse d'un joueur adverse), ou une creuse (si la dernière case contient une pleine d'un différent joueur) ou si encore aucun choix de pions n'est possible, indiqués au joueur de jouer sur une autre colonne.

Maintenant que nous pouvons placer des pions sur le plateau de jeu, il nous faut une fonction qui permet de vérifier s'il y a une combinaison gagnante (4 pions alignés horizontalement, verticalement ou en diagonale).

Pour commencer nous avons créer une seule et unique fonction qui vérifier toutes les cases de la grille.

Après plusieurs heures passées dessus nous avons du reconsidérer le problème... Nous avons tout repris à zéro !

En effet, pour pouvoir résoudre ce problème nous avons du comprendre qu'il fallait étudier la où le joueur vient de jouer, et non toutes les cases de la matrice.

C'est à dire qu'à chaque fois que le joueur place un pion on parcourt toutes les directions qui entourent la dernière case jouée.

Nous avons du créer une fonction 'case_du_joueur' qui teste la case et renvoie vraie s'il y a un pion de la couleur du joueur, faux sinon.

A partir de là nous avons pu facilement créer une fonctions booléenne 'victoire', qui fait appel à 4 autres fonction booléennes 'horizontale_win', 'verticale_win', 'right_diagonal_win' et 'left_diagonal_win'.

Plus précisément :

On positionne un indice de la ligne de la case et si on rencontre un pion de la même couleur on continue horizontalement vers le haut sinon à l'inverse repartir de la case jouer par l'utilisateur et effectuer le parcours.

Si horizontalement le pion est de la même couleur lors du parcours des cases, on continue dans cette direction et on incrémente un compteur sinon on change de direction et on remet le compteur à zéro. Si le compteur est égale à 4 ou plus alors il y a 4 pions alignés !

Après quelques heures d'écriture notre fonction 'victoire' était fin prête et opérationnelle.

Nous sommes ensuite passé à la gestion du score des joueurs.

Pour toute cette partie nous avons décidé de stocker le informations sur le joueur dans un fichier texte. Pour chaque joueur on aura son nom, son score (nombre de partie gagnée) et son nombre total de parties jouées. Afin de pouvoir travailler avec ses informations il nous fallait récupérer les données du fichier et les stockées dans une variable. Dans ce contexte nous avons décidé de créer une structure t_joueur composé de 3 champs nom, score et total, et un tableau de structure t_joueur tabjoueur[J] où l'on stockera les données contenu dans le fichier.

Maintenant que nous pouvons jouer une partie et gérer les joueurs, nous nous sommes occupés du côté plus « esthétique » du programme. C'est à dire que nous avons créé un petit menu de départ qui demande le nombre de joueur souhaitant jouer:



image 1 - Premier menu du jeu Puissance 4++



image 2 - Second menu du jeu Puissance 4++



image 3 - Dernier menu pour les paramètres du joueur

III/ MISE EN PLACE DE L'ASPECT VISUEL

a) Le puissance 4 est jeu composé d'une grille de 6 lignes et 7 colonnes, avec 1 seul type de pion classique, soit un jeton qui se place au dessus d'un autre pion déjà dans la même colonne choisie, et seulement 2 joueurs peuvent jouer l'un contre l'autre.

Notre projet, soit le Puissance 4++, modifie les règles basiques de la version précédente, on aura ici 3 types de pions et jusqu'à 4 joueurs pouvant s'affronter, et c'est là que l'aspect graphique est important, afin de différencier les types de pion et le pion de chaque joueur, ainsi un meilleur visuel sur l'ensemble du jeu.

b)Voilà ci-dessous, la grille Puissance 4++, avec un cadre pour améliorer son aspect, sur l'image on peut voir trois un espacement entre chaque caractère, à chaque colonne, ceux sont les espaces vides de notre structure de base qui représentent les 3 types de pions(bloquant, creux et plein), on a choisit tout simplement cette façon de faire afin d'avoir des colonnes vides et nettes.



image 4 - Grille de jeu avec victoire pions noirs

c) Sur l'image, on peut observer les 3 types de pions : (ici 4 joueurs s'affrontent)

P : pour les pions Plein

C : pour les pions Creux

B : pour les pions bloquants

d) On peut observer jusqu'à 4 couleurs différentes, qui représentent chaque joueur (Le Noir, Le Cyan, le vert et Le Blanc).

On constate donc un meilleur aspect, grâce aux couleurs, où on peut distinguer les différents pions de chaque joueur.

e) Pour remplacer les « VIDE », par des espaces vides, on a seulement utilisé un « define VIDE '' », et pour l'encadrement et la numérotation des colonnes, on a réuni cette partie dans un « void afficher_grille(Pion grille[N][M]) ». Plus explicitement, on a utilisé des boucles « for », qui affichent les numéros de chaque colonne et les contours de la grille, pour la couleur de chaque joueur et les types de pions, nous avons initialisé une variable « coul_joueur » qui prendra en image un numéro, qui sera compris entre 1 et 4, qui correspondra à l'une des 4 couleurs citées dans les parties précédentes, et les types de pions seront affichés selon ce que l'utilisateur notera au clavier.

On a donc défini par exemple : une fonction « couleur(codage(joueur[0].couleur) », pour distinguer les pions de chaque joueur.

f) Les problèmes rencontrés étaient au début pour les couleurs, il y avait avant tout, une couleur pour chaque type de pion, un système logique pour deux personnes, ensuite, on a augmenté le nombre de joueur, et il nous a fallu qu'au lieu d'incrémenter un simple chiffre qui représentera un joueur, on donne une image à ce chiffre, qui sera une couleur donnée et gardée par chaque joueur jusqu'à relancer une nouvelle partie.

IV/ Tests et problèmes

a) Les limites, problèmes et Solutions

Comme tous projet, il y a des limites, et comme tous programme, même si l'innovation est très très vaste, surtout dans le monde de l'informatique, on a eu ici nos limites et problèmes.

En C, on ne dispose pas énormément de couleur, bien qu'on aurait pu rajouter 2 ou 3 autres couleurs, soient 3 autres joueurs en plus, mais on avait décider directement en début de projet, à limiter à 4 jours, du fait qu'on dispose 3 types de pions different et une grille plutôt restreinte pour plus de joueur.

On étaient aussi partis avec l'idée de créer un tableau en fonction du nombre de joueur choisit par l'utilisateur, pour sortir un peu de l'ordinaire.

Finalement, on a gardé cette idée en tête, mais on voulaient tout d'abord avoir un programme fonctionnel .

La fonction qui nous a donné le plus de problème est la fonction victoire, plus précisément au niveau de la diagonale, et donc aussi la fonction aide qui partait du même principe.

En effet, pour pouvoir résoudre ce problème nous avons dû comprendre qu'il fallait étudier chaque case de la matrice et non pas seulement partir de la case où le joueur vient de jouer. C'est à dire qu'à chaque fois que le joueur place un pion on parcourt toute la grille à la recherche d'une combinaison de 4 pions alignés.

Enfin, la dernière fonction qui nous a aussi posée quelques difficultés, était la fonction « help_me », qui permettait de demander une aide à l'ordinateur, après bonne réflexion, on a tout simplement utilisé le même système que la fonction de détection de victoire.

C'est-à-dire que notre fonction cherche case par case des pions alignés et joue donc au-dessus de celle-ci.

V/ Conclusion

En conclusion, nous avons beaucoup appris grâce à ce projet !

Dans un premier temps, nous avons appris lors des différentes séances et temps de vacances, à gérer un projet avec ses contraintes, bien respecter le cahier des charges et le planning.

Ensuite, nous avons remarquer, le plus difficile n'était pas enfaite la conception, ni finir le code avant les temps, mais le plus difficile pour nous, durant cette expérience était surtout au niveau du travail d'équipe, on a appris qu'une bonne organisation est a base d'un projet, que tous les membres du groupe doivent être sur le même optique. C'est donc après ces épreuves que nous avons su surmonter cela et créer un programme fonctionnel.

Nous sommes satisfait de notre résultat final, même si comme tout programme, celui ci peut être encore très largement amélioré !