PROGRAMMATION C : PROJET PICKOMINO

RAPPORT

BITTEL Axel – BETBETDER Julie

Introduction :

Passer de la théorie à la pratique est indispensable, d'autant plus lorsqu'il est question de programmation : aller chercher l'information, se confronter aux erreurs de compilation, imaginer des structures, une architecture de code, traduire les problèmes posés… Il suffit de quelques secondes pour s'en persuader et de quelques lignes de code pour finir de s'en convaincre.

C'est dans cet état d'esprit que nous avons commencé ce projet. Nous y avons vu l'occasion de s'améliorer sur de nombreux points (tels que le travail en groupe, le langage C en lui-même, les manières d'appréhender un sujet et de le traduire…) tout en prenant plaisir à voir notre avancée, notre aise croissante ainsi que l'évolution de nos connaissances.

I) Une brève présentation

A) De notre binôme…

L'idée de travailler en binôme nous est venue assez rapidement, tout comme notre association. Nous sommes tous deux en double licence mathématiques-informatique et notre temps libre est assez limité (école 42 pour l'un, conservatoire pour l'autre). Ainsi, nos horaires de travail communs se concentraient principalement en soirée et, afin d'avancer séparément, nous sommes également passés par GitHub.
Notre duo a plutôt bien marché. Nous sommes avant tout deux amis, il a donc été assez facile de se comprendre et de se mettre d'accord sur la manière de gérer le projet.

De plus, notre différence de niveau nous a été bénéfique à chacun.
En effet, Axel est bien plus expérimenté dans ce domaine. Il s'est plongé dans ce monde de programmation des années en arrière et s'y trouve comme un poisson dans l'eau, particulièrement lorsqu'il est question de langage C. En revanche, Julie patauge depuis et peine déjà légèrement à garder la tête hors de l'eau.
Des efforts ont été faits des deux côtés afin que personne ne se sente à l'écart : Julie a beaucoup appris, elle a pu poser des questions, se faire guider et participer activement ; Axel s'est mis au défi de simplifier au maximum ses explications et ses lignes de code afin de l'aider et la soutenir de son mieux, un exercice loin d'être simple auquel il a su répondre parfaitement.

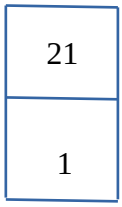
Finalement, le travail s'est effectué ensemble jusqu'au bout et nous a demandé une certaine rigueur en ce qui concerne les dates de mise en commun etc..

B) Et du projet

Il était ici question d'implémenter le jeu pickomino en C afin de pouvoir y jouer en mode console. Nous voulions répondre au plus près possible des consignes données.
Pour se faire, une notion de menu/sauvegarde a été écrite, tout comme la possibilité de jouer au minimum à deux, au maximum à 7 et ce avec le nombre de joueurs autonomes voulus (notamment 0 si on ne joue qu'avec des personnes « physiques », et 7 si on ne fait jouer que des robots).

Le jeu se compose de 8 dés à six faces (cinq portent les valeurs de 1 à 5 et la dernière porte la valeur V, qui vaut 5 points) ainsi que de 16 pickominos (représentés par des rectangles divisés en deux parties : en haut une valeur entre 21 et 36 ; en bas une valeur entre 1 et 4).

Nos représentations physiques :

Pickomino : 

Face d'un dé : [1]

Afin de rendre le jeu plus agréable, il a également été décidé d'afficher un substitut de table et de tapis de jeu qui sont figés. Seul leur contenu peut être modifié en fonction de l'évolution du jeu, ce qui apporte une certaine constance et rend l'ensemble plus propre.
Ainsi, les questions guides ainsi que les lancers de dés se trouvent en dessous, et apparaissent et disparaissent lorsqu'il le faut.

Rentrons plus précisément dans le jeu, et survolons les grands principes.
Au début de la partie, les 16 pickominos sont disposés face visible sur le centre de la table ; le but étant de les récupérer et d'obtenir le plus de points possible (en additionnant les valeurs de la partie inférieure).
Pour récupérer un pickomino, on démarre un tour (le joueur A joue). Il faut dans un premier temps lancer les dès, à lui ensuite d'établir sa stratégie : il doit garder certains dés et relancer les autres autant de fois que voulu (bien qu'il ne puisse pas garder deux fois la même valeur, qu'il est obligé de garder au moins un dé par lancer -son tour est considéré comme raté s'il ne peut récupérer aucun dé lors d'un lancer - , et qu'il a besoin d'au moins un V s'il veut pouvoir prendre un pickomino).
Le joueur A décide de ne plus lancer les dés.
S'il a gardé au moins un V, alors il faut comparer son score (on additionne les valeurs des dés gardés, en considérant que la face V vaut en 5 points dans ce cas) aux valeurs supérieures des pickominos. Si le score obtenu est supérieur ou égal à la valeur d'un pickomino sur la table, alors il récupère le pickomino dont il est question et le place sur une pile devant lui. Il est important de noter que les pickominos récupérés sont empilés les uns sur les autres, seul le dernier récupéré par un joueur est donc visible. Il est possible de prendre le pickomino visible d'un autre joueur si la valeur dudit pickomino est exactement égale au score du joueur.

Le tour se termine alors, et le joueur suivant peut commencer le sien.
Lorsqu'il n'y a plus aucun pickomino sur la table, le jeu s'arrête et le gagnant est désigné.

II) Notre logique de travail

A) L'architecture du code (la forme)

Avant même d'écrire notre première ligne de code, un gros travail de réflexion a été fourni à l'oral et sur papier. Il nous paraissait fondamental de s'imprégner de la logique du jeu pour mieux comprendre comment l'aborder et enfin le coder. Nous en avons discuter, nous y avons jouer et nous en avons déduit les étapes principales qui allaient structurer et diviser notre code.

On remarque assez facilement (surtout dans le fichier jeux.h) que notre code se divise en 8 grandes parties : les fonctions en lien avec l'affichage (1), celles pour gérer l'entrée des données (2), celles qui gèrent les joueurs et le plateau (3), celles qui concernent les dés (4), celles pour les joueurs autonomes (5), celles sur la logique du jeu à proprement parler (6), celles pour la sauvegarde (7) et enfin un petit paquet de fonctions utiles et usitées à plusieurs reprises dans de nombreuses autres fonctions (8).

Au delà de ça, nous avons travaillé sur deux structures principales : une structure joueur (pour chacun, on y trouve son nom, sa qualité de personne « physique » ou de bot, ainsi que les pickomino qu'il a récupéré) et une structure data (qui regroupe les données du jeu avec le nom de la partie et le nombre de joueurs qui y participent, la valeur des dés à chaque lancer et les pickominos sur la table). En outre, cette deuxième structure pointe vers la première.

A l'intérieur de ces grandes parties, plusieurs fonctions ont été écrites.
Pour plus de lisibilité, nous avons essayé de nous limiter à des fonctions assez courtes (en général une vingtaine de lignes) et assez simples dans leur fonctionnement (en évitant d'imbriquer trop de boucles ou de conditions), et nous avons usité des noms de variables et de fonctions assez parlants ainsi que des commentaires pour pouvoir reprendre notre code plus facilement.
Ces règles nous ont également permis de rendre le travail séparé plus simple : nous comprenions plus facilement ce que l'autre avait voulu faire et ainsi il nous pouvions continuer le code ou reprendre les fonctions écrites par l'autre.

B) Les stratégies et fonctions principales (le fond)

Détaillons maintenant ce qui se trouve dans chacune des parties citées plus haut. Nous en profiterons pour donner la stratégie concédée aux joueurs autonomes.

L'une des parties principales de notre code, qui a d'ailleurs mise à rude épreuve notre patience a été l'affichage. Nous avons tenté de le rendre le plus agréable possible mais le stockage et la gestion nous ont posé quelques problèmes. Nous nous sommes alors inspirés du traitement d'image : l'idée est de créer un tableau à deux dimensions de long long (= 64 bits) afin de pouvoir stocker les caractères spéciaux dits multi-caractères et d'afficher le résultat final avec print_screen. Ainsi on sépare chaque long long en octets qu'on affiche alors séparément.(1)

En ce qui concerne l'initialisation de nos structures, nous avons consacrées une fonction par structure, rangées dans la partie « data management » (init_joueur et init_data).
Cette partie comporte également les fonctions d'action ou de recherche en ce qui concerne les pickominos des joueurs et ceux présents sur le plateau. (2) (3) Là encore, par soucis de lisibilité, nous avons décidé d'en écrire plusieurs qui se concentrent sur une action précise.
Dans la continuité, il paraît logique de s'occuper maintenant des dés. Quelques petites fonctions intermédiaires ont encore une fois été écrites pour simplifier le code de la fonction principale : score_des .

On peut diviser la logique de jeu en elle-même en deux grandes parties : la gestion des dés et la gestion des pickominos. Une fois les fonctions sur ces deux étapes écrites, il ne reste plus qu'à les mettre en lien. C'est la partie logique et gestion du jeu qui s'en charge et notamment la fonction update_game (6).

Enfin pour clore le jeu proprement, il a été question de désigner le vainqueur et d'afficher les résultats de tous. Pour lui donner une dimension plus finie et aboutie, nous nous sommes occupés de la sauvegarde. Il est ainsi possible de reprendre une partie déjà commencée en donnant son nom ou, si la partie dont il est question a été terminée, d'en afficher les résultats (7).

Une autre partie a été traitée un peu à part : celle qui concerne les joueurs autonomes (5). Nous avons voulu leur donner une stratégie de jeu qui prend en compte les pickominos sur le plateau. Ainsi, pour éviter le tour raté, on décide de leur faire prendre un V dès qu'ils en ont l'occasion. S'ils en ont déjà récupéré un, ou s'ils n'en ont pas dans leur lancer, on essaye de leur faire faire le maximum de points. Pour ne pas leur faire perdre trop de dés d'un coup, si la valeur choisie est inférieure ou égale à 3, on limite le nombre de dés qu'ils peuvent prendre. A moins qu'un lancer où ils ne peuvent rien prendre ne les arrête, on fait tourner cette stratégie tant que leur score n'est pas supérieur ou égal à la valeur minimale des pickominos prenables sur la table.