Rapport du projet unix de L2 : Puissance 4

Martin Prieur / Abel Derderian

17/12/2014

Table des matières :

1) Introduction au projet

Dans le cadre de nos cours d'unix, nous avons appris à utiliser la ligne de commande en commençant à la base. Fonctions, commandes, arguments, arborescences et système de fichier unix.

Après plusieurs travaux pratiques nous demandant de trouver les meilleures commandes possibles pour avoir des résultats en fonction de l'énoncé, nous avons découvert l'utilisation de création de fichiers de scripts : les fichiers shell (ou .sh).bn

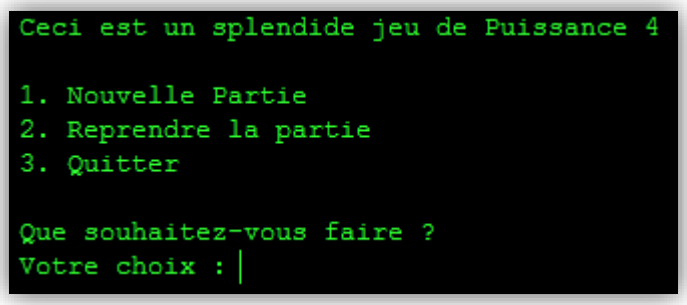
Le projet que nous avons eu demandait la création du jeu puissance 4 en script shell, les conditions requises étaient les suivantes :

1) **Création d'un menu de sélection**
   - *Nouvelle partie*
   - *Charger partie*
   - *Quitter partie*
2) **Possibilité de sauvegarde / chargement de partie**
3) **Possibilité de quitter le jeu.**
4) **Possibilité de jouer au jeu avec gestion des tours, des fins de parties et évidemment des cas de « Puissance 4 ! ».**

Nous détaillerons chaque fonctionnalité par la suite que ce soir de manière algorithmique ou en termes de programmation.

1) Présentation globale de l'algorithme

Dès l'ouverture du script, on va commencer par arriver sur un menu avec une boucle à condition qui encadrera l'ensemble du script. Un choix est demandé dès le début afin de déterminer la suite du déroulement de l'algorithme.



Avec un simple menu, l'utilisateur sélectionne ce qu'il souhaite faire, et l'algorithme redirigera l'utilisateur dans la fonctionnalité correspondant à sa demande.

Dans l'éventualité la plus simple où le joueur souhaite quitter le jeu, le programme s'arrête simplement et effectue un nettoyage de la console (pour plus de « propreté »).

Si le joueur souhaite charger une partie, le programme appellera alors la fonctionnalité de chargement / sauvegarde, qui consiste à lire les données écrites dans un fichier texte adjacent qui contiendra les données du jeu. A savoir l'état du plateau, le joueur en cours et son symbole (le jeton) et reprendra (ou commencera) une partie avec les valeurs obtenues.

La partie la plus importante correspond évidemment à la nouvelle partie, car c'est celle-ci qui nous a demandé le plus de travail. Si l'utilisateur souhaite faire une nouvelle partie, le script va alors générer un tableau de 6 lignes par 7 colonnes. Chaque joueur pourra sélectionner chacun à son tour une colonne de son choix, dans lequel un pion ayant le symbole du jeton du joueur actuel défilera. A chaque fin de tour, l'algorithme vérifie s'il n'y a pas la présence de puissance 4 avec la nouvelle pièce ajouté, par comparaison des différentes combinaisons de pièces, et si ce n'est pas le cas ça sera au tour du joueur suivant. Une option avec la touche S permet également de sauvegarder le plateau.

La sauvegarde va prendre la position de chaque pièce pour la stocker dans un fichier texte, pour une utilisation future.

2) Explication des fonctionnalités

Afin d'optimiser notre code et de pouvoir développer de manière plus facile, nous avons sélectionnés les fonctionnalités que nous voulions pour en faire des fonctions utilisable de manière globalement toutes indépendantes.

Nous allons détailler l'algorithme de manière plus informatique, et expliciter les différentes fonctions codés.

Nous avions commencé un menu graphique avec DIALOG pour favoriser l'ergonomie du jeu mais depuis quelques semaines, l'environnement d'Efrei ne le permet plus (comptabilité, mise à jour ou autre raison). Nous sommes donc repassés sur un menu normal. Dès le début, l'algorithme entre dans une boucle pseudo-infinie : la condition ne se termine pas et c'est ce qui encadrera l'ensemble de l'algorithme. On affiche le menu, et suite à ça, on entre dans une seconde boucle où l'utilisateur choisira ce qu'il souhaite faire.

En fonction de son choix, l'algorithme appellera 2 fonctions externes : **New_game** ou **Chargement**.

L'option Quitter tient simplement à la sortie du programme et au nettoyage de l'écran.

Chargement : L'algorithme va générer un tableau, puis ouvrir le fichier Sauvegarde.txt, qui doit être adjacent au programme, lire ce fichier lire par ligne et remplir en conséquence le tableau. On obtient donc le tableau que l'utilisateur avait sauvegardé au départ. Egalement, on note le joueur actuel et son jeton pour que le jeu reprenne au tour adéquat.

New_Game : La fonction « principale » du programme.

Dès le début de cette fonction, le programme va générer un tableau de 6 lignes par 7 colonnes (même si en réalité le tableau contient une colonne supplémentaire nécessaire à l'affichage des indices sur le côté latéral gauche).

Dès que le tableau est affiché (ceci se fait via une fonction externe, afin de pouvoir le refaire à chaque tour), c'est au tour du joueur 1 de commencer. On attribue au joueur 1 le symbole X et au joueur 2 le symbole O.

Lorsque leur tour arrive, chaque joueur est invité à donner un nombre entre 1 et 7, correspondant à la colonne dans laquelle ils souhaitent joueur. Si la sélection n'est pas strictement égale à 1, 2, 3, 4, 5, 6, 7 ou S (fonction de sauvegarde), le programme redemande à l'utilisateur de faire une sélection correcte. Le jeu ne se poursuit pas tant que l'utilisateur n'a pas donnée une valeur attendue.

Quand l'utilisateur a choisi sa colonne, le symbole défile dans la colonne précisée jusqu'à atteindre le bas. En réalité, l'algorithme vérifie si la case la plus haute de cette colonne est vide : si elle ne l'est pas, il affiche un message d'erreur à l'utilisateur (qui choisira une autre colonne). Si elle est vide, il place un pion dans cette colonne, patiente un temps d'attende de l'ordre du dixième de seconde, vérifie si la colonne en dessous est vide et descend le pion si cela lui est possible. Le jeton s'arrête donc au plus bas possible.

Sauvegarde : Egalement, le joueur pourrait taper S et effectuer une sauvegarde de la partie. Cela aurait pour effet de parcourir l'ensemble du tableau et note les différentes cases présentes dedans pour les écrire dans un fichier texte. Le joueur actuel ainsi que son jeton est également sauvegardé pour que la partie reprenne là où elle s'est arrêtée.

A la fin de chaque tour, l'algorithme va analyser pièce par pièce le plateau de jeu afin de déterminer si la combinaison des pièces présentent permet une victoire (puissance 4 !) pour l'un des deux joueurs. On effectuera une vérification en fonction de chaque pièce en analysant les pièces à sa proximité :

- Diagonale basse : droite et gauche.
- Côté horizontal : droit.
- Côté vertical : bas.

Nous n'effectuons pas de vérification pour la diagonale haute étant donné que nous parton de la case la plus à gauche en haut de la console. Par conséquent, chaque pièce pouvant faire un potentiel puissance 4 diagonal-haut aura déjà été vérifié par l'analyse du diagonale-bas du jetons supérieur de 4 ligne.

Si à la fin de cette analyse, on observe qu'une pièce dispose d'un côté comprenant 3 jetons similaires, on déclare le puissance 4 pour le joueur.

Verif_plateauplein : Une autre vérification en fin de tour est de s'assurer que le plateau n'est pas plein. Si la première ligne contient au moins une valeur correspondant à « . » (symbole d'une case vide), alors le plateau n'est pas plein. Si on ne trouve pas cette valeur, le jeu prend fin puisque le plateau est rempli.

Si toute les vérifications ont été passées, que le plateau n'est pas plein et que le puissance 4 n'est pas encore pour ce tour, le stratège adverse commence son tour, et le jeu continue jusqu'à ce qu'il y a ait une victoire.

Conclusion

Ce projet nous a permis d'apprendre l'utilisation d'un script shell et les différentes spécificités qu'il peut avoir par rapport au C, C++ ou PHP (qui contiennent les mêmes conditions de base (IF, WHILE, FOR)) que ça soit au niveau de la syntaxe ou de la gestion des variables.