MORPION

Projet 1 en langage C

Laduranti Sandra
L2-ATIC
UPMC

Programmation en C

14 Octobre 2013

Sandra.ladu@etu.upmc

Table des matières

I. Introduction

1. Projet

Le but du projet est de programmer un morpion avec une grille dynamique et gestion d'options multiples tel que le nombre de joueurs.

La version v4 doit pouvoir gérer une IA. Le programme proposé ici s'arrête à la version précédente.

Au travers de ce projet les objectifs à atteindre sont la maitrise des structures, des types et une meilleure compréhension des mécanismes de la mémoire.

2. Technologie(s) utilisée(s)

Dans le cadre de ce projet le programme a intégralement été développé en C avec utilisation des libraires *stdlib, stdio*, *unistd* et *getopt*.

3. Cadre du projet

Ce projet est réalisé dans le cadre du cours de programmation en langage C.

II. Développement du projet

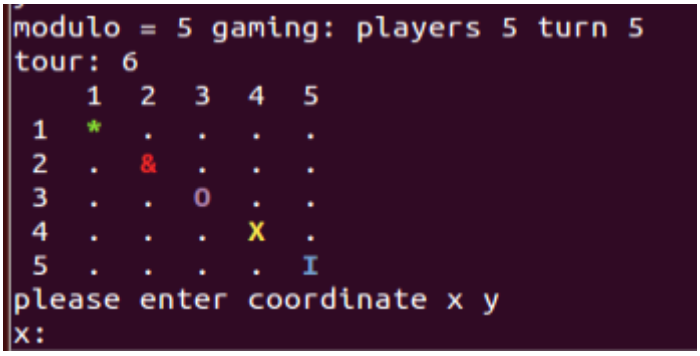
1. Capacités actuelles du programme

Afin de ne pas imposer à l'utilisateur, qui voudrait jouer de manière classique, un grand nombre de questions concernant les options avec lesquelles celui-ci souhaiterait jouer.
J'ai donc fait le choix d'utiliser les arguments du *main* afin de rentrer directement toutes les options que le joueur voudrait utiliser.
Il existe actuellement 4 options dont 3 sont déjà implémentées dans le code :

```
patate@patate-VirtualBox:~/Desktop/morpion$ ./morpion -h
WELCOME ON MORPION 1.0!
If you want options please use ./morpion -h
Usage:morpion[options]
 Options: --n correspond à un entier décimal--
          -gn Définit la taille n de la grille
           par défaut n = 5, la taille ne peut pas être inférieure à 5
          -jn Définit le nombre n de joueurs
           par défaut n = 1, il ne peut pas y avoir plus de 5 joueurs
          -ln Définit le nombre n possible d'utilisations d'une même case
           par défaut n = 1
    1  2  3  4  5
 1  .  .  .  .  .
 2  .  .  .  .  .
 3  .  .  .  .  .
 4  .  .  .  .  .
 5  .  .  .  .  .
please enter coordinate x y
x:
```

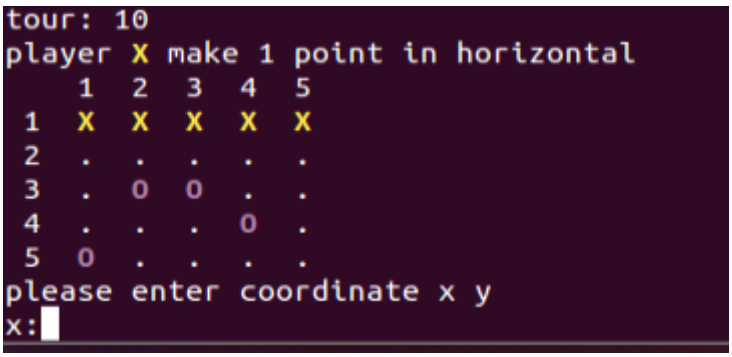
Il est possible de rentrer l'option du nombre d'utilisation de la case unique par ligne qui est enregistré dans une structure mais pour le moment n'est pas exploité au niveau de l'algorithme.
Dans le cas où les options ne sont pas rentrées alors les paramètres de base sont placés à 2 joueurs (-si une personne souhaites jouer seule il y aura automatiquement un changement de couleur et donc pas de différence au niveau du code par rapport à deux joueurs-), il n'est pas possible de dépasser 5 joueurs. Dans le cas où l'utilisateur tente de rentrer un nombre plus

grand de joueurs le programme considère qu'il y a eu 5 joueurs de rentrés. Pour faciliter le jeu j'ai choisi de mettre en couleur l'affichage des joueurs.

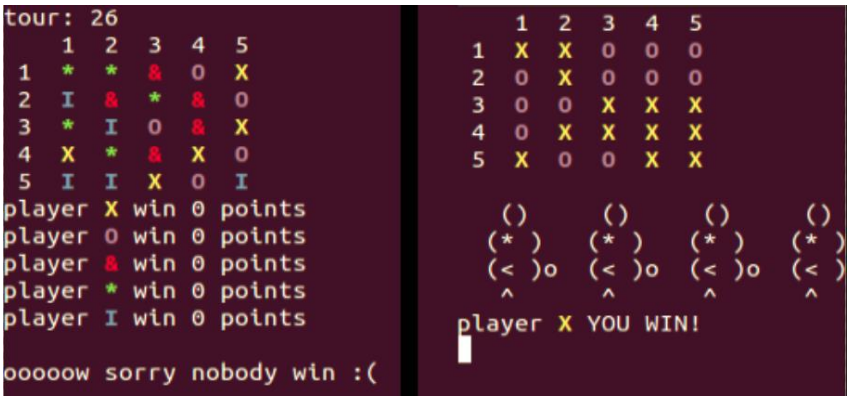


La grille de jeu est limitée au minimum à 5, si l'utilisateur tente de rentrer un nombre inférieur à 5 alors le programme ignore la demande et prend en compte la valeur par défaut. Il est possible de mettre une grille supérieure à mais il y a cependant des soucis au niveau de l'affichage qui se décale légèrement lorsque l'on atteint la dizaine.

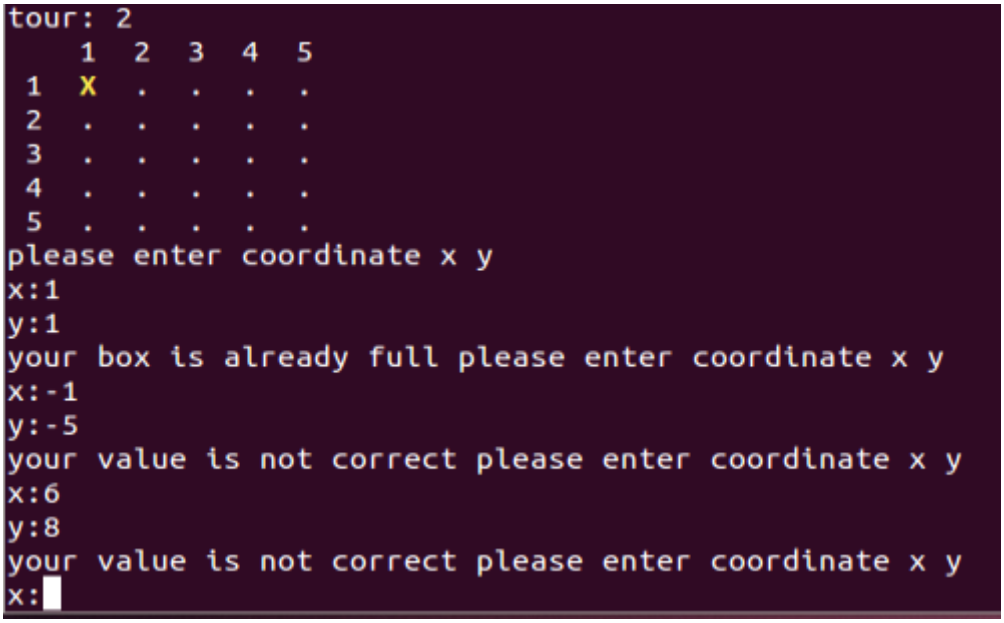
Le programme calcul si un joueur a gagné un point ou non et affiche dans le cas où il aurait marqué un point si il a été gagné en horizontal, vertical, diagonale droite ou diagonale gauche.



A la fin du jeu le programme affiche match nul dans le cas où aucun point n'aurait été marqué et affiche qui a gagné dans le cas où au moins un joueur a marqué un point. Une boucle vérifie que le gagnant (donc le plus grand nombre de points) est unique, si ce n'est pas le cas elle renvoie sur l'affichage du match nul.



Le programme vérifie également que la valeur de jeu entrée par l'utilisateur est correcte, elle doit-être comprise dans la grille. SI c'est le cas alors le programme check si la case demandée est déjà pleine ou si elle est disponible.
Tant que l'utilisateur n'as pas rentré de valeur correcte le programme tourne en boucle en redemandant une nouvelle valeur.



2. Choix de librairies

Les librairies classiques du C tel que *stdlib* et *stdio* ont été ajoutées automatiquement à chaque fichier afin de pouvoir utiliser le *printf* pour l'affichage classique, le *fprintf* avec *stderr* pour la sortie d'erreur, le *malloc* pour l'allocation mémoire, le free pour libérer la mémoire, l'*atoi* pour la conversion de *int*, etc...
J'ai également choisi d'utiliser *unistd* afin de pouvoir utiliser le write et usleep pour faire un affichage en mouvement du gagnant.
Et enfin ce projet a été l'occasion de découvrir la librairie *getopt* et ses mécanismes.
J'ai utilisé cette librairie pour la récupération et le parsing automatique des arguments pris par le *main* et donc ainsi vérifier leur existence mais également les insérer dans une structure au fur et à mesure.

3. Algorithme général

Le programme prend d'abord en compte l'existence d'options s'il y en a –grace à l'utilisation de getopt qui parse les commandes entrées en argument du main-, dans ce cas il remplit la structure correspondante avec les infos rentrées si jamais elles rentrent dans les règles données du morpion.
Le programme récupère la valeur, imposée à 5 s'il n'y a pas eu de précisions dans les options, corrigée de la grille et créée un tableau dynamique de *int* avec un *malloc*.

La grille est entièrement initialisée avec la valeur VIDE de type FAUX correspondant à la valeur 0.
Durant tout le reste du programme le tableau sera transmis entre les fonctions par pointeurs, de même que pour la structure *t_game* contenant les options et le nombre de tours joués.
Le programme est composé d'une boucle vérifiant le retour de fonction de FINISH qui arrive lorsque la grille a été intégralement remplie.
A chaque tour de boucle le programme demande la valeur de case à jouer, vérifie son existence et sa disponibilité.
Dans le cas où tout est bon alors la case est remplacée par le numéro de joueur correspondant au tour de jeu. La difficulté ici fut de trouver comment déterminer à quel joueur correspondait le tour de jeu.
Si jamais la case n'existe pas (inférieure ou supérieure à la taille de la grille) alors le programme reste bloqué dans la fonction récupérant les coordonnées de la case tant qu'aucune valeur correcte n'as été rentrée.

Une fois la case validée le programme part dans la vérification de l'alignement de 5 cases de manière soit horizontale, soit verticale, soit diagonale droite soit diagonale gauche. Dans le cas où 5 cases sont alignées alors le joueur auquel correspond la couleur gagne un point, la valeur est enregistrée dans un tableau situé dans la structure *t_game*.

Le programme reboucle au début et revérifie donc si toutes les cases de la grille ont été jouées. Si c'est le cas il affiche les scores et détermine qui a obtenu le score le plus haut pour afficher un gagnant.

Dans le cas de l'utilisation de l'option –c la boucle est sensiblement différente, la limite de joueur se trouve à 2 et dans ce cas la gestion du tour de jeu se fait avec un modulo 2. Le joueur gardera le même schéma algorithmique que vu précédemment, lorsque le tour de l'ordinateur arrive il utilise une fonction qui détermine quelle case il peut jouer en random.
On initialise préalablement srand avec time.
Une boucle check si la valeur random trouvée se situe dans la grille puis si la case est déjà remplie, dans le cas où la case est disponible le programme retourne dans les mêmes fonctions de remplissage que celles utilisées pour le joueur.

III. Conclusion

Il a été difficile de programmer de manière à pouvoir changer les règles au fur et à mesure du projet.

Une erreur au niveau de l'algorithme random m'as fait perdre énormément de temps, à savoir une inversion de x et y dans le remplissage de tableau qui menait à la réécriture sur des cases déjà remplies.

Dans la première version de l'ia prévue j'avais prévu un algorithme min-max sur un arbre, afin de mettre des niveaux de jeux il était prévu de donner sa construction en 3 niveaux de profondeur différents selon la demande du joueur.

Afin de ne pas reproduire tout le temps le même schéma de jeu le min-max devait calculer, dans le cas où il y aurait au moins deux évaluations égales sur un niveau, un random entre les deux branches concernées.

Il s'est avéré que la fonction d'évaluation était plus complexe que prévu et le second choix d'algorithme s'est porté sur un random défensif qui aurait réutilisé les fonctions calculant combien de points ont été alignés. Dans le cas de 4 points alignés 'libres' donc non compris les bords de la grille ou un autre joueur ayant joué à côté, l'ia devrait placer sur un des deux emplacements son pion. De même que dans le cas de 3 points alignés. Dans le cas où l'ia ne trouve plus, ou alors moins de 3 points alignés elle joue en random.

Eventuellement ajouter un calcul pour tenter de faire marquer des points dans ces cas-là à l'ia aurait été intéressant. Cependant l'erreur précédente ne m'as pas permis de finaliser cette version de l'ia.

Lors de ce projet j'ai pu découvrir la librairie optget qui m'as fait gagner énormément de temps sur le parsing que j'avais prévu, au début, de faire à la main.