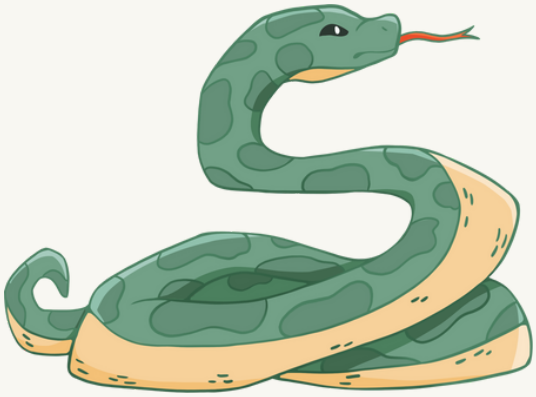
Notice jeu snake



DJEDJIGA YAHIAOUI

MARWA HAFSATI

INTRODUCTION

Bienvenue dans le notice du jeu Snake. Ce document vous fournit toutes les informations nécessaires pour comprendre notre programme en langage C89, permettant ainsi, la création de ce jeu.

Le jeu consiste à déplacer un serpent à l'aide des touches directionnelles pour qu'il puisse se nourrir de pommes, augmentant ainsi sa taille.

/!\ Attention, il faut éviter que la tête du serpent ne touche les parois ou une partie de son propre corps, car cela entraînerait la fin de la partie. De plus, appuyer sur la <**barre d'espace**> met le jeu en pause et appuyer sur la touche <**ESC**> permet de quitter le jeu.

5 STEP PROCESS

01 MAKE FILE

02 LE FICHIER MAIN.C

03 GESTION_SERPENT.C

04 GESTION_CHRONO

05 LA VARIANTE

LE MAKE FILE

Dans notre Makefile, nous avons associé différents fichiers objet (.o) à leurs fichiers sources (.c) et en-têtes (.h) correspondant. Ce fichier nous permet d'organiser nos fichiers de manière à rendre notre code plus lisible, compréhensible et fonctionnel. Nous avons donc créé un fichier point c et point h dédié à:

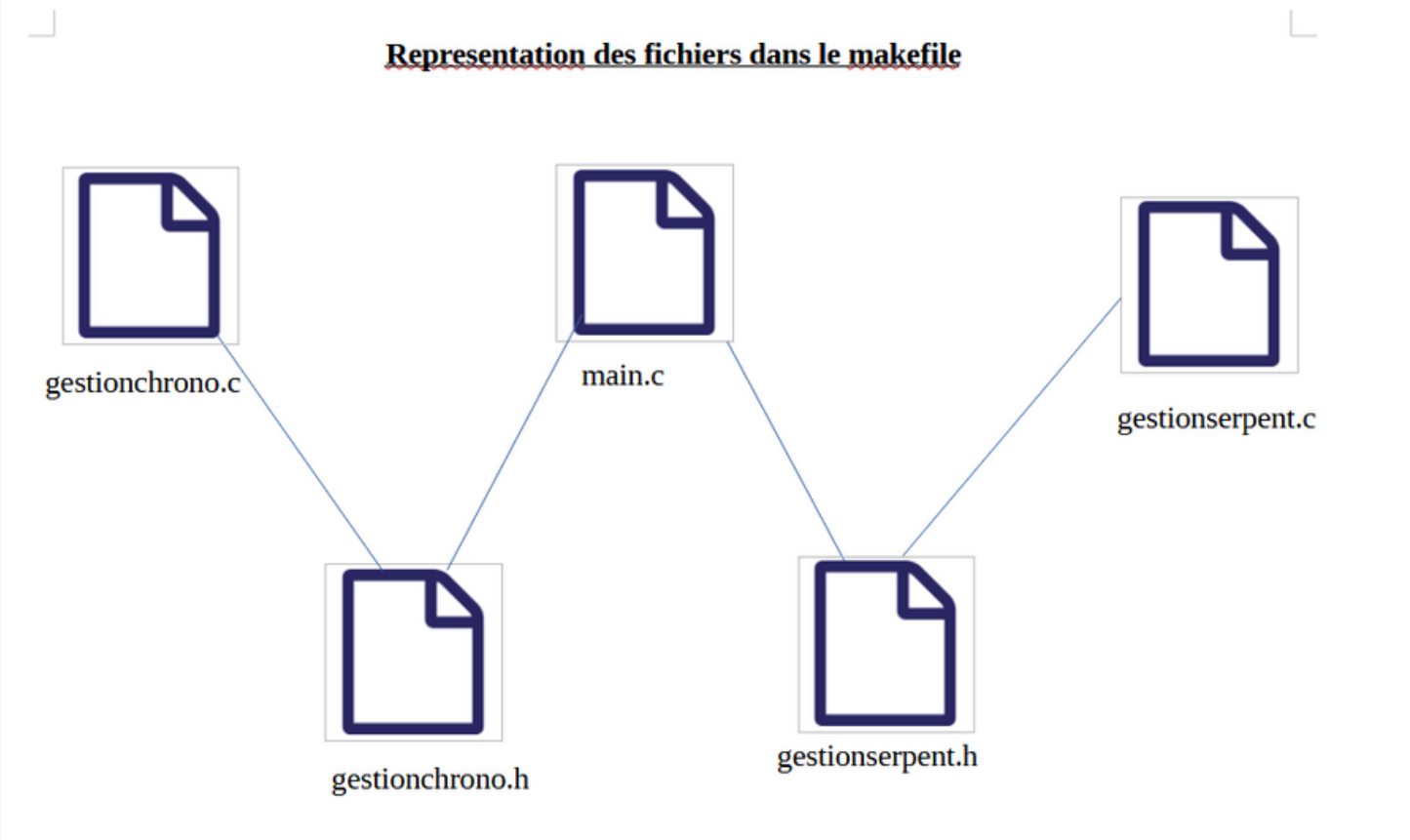
- **La gestion du serpent** et de **ses collisions** avec les **pommes**(gestion_serpent.c/gestion_serpent.h)
- **La gestion du temps** et **des pauses** (gestion_chrono.c/gestion_chrono.h).

Mais aussi un fichier source **main.c** qui contient notre **boucle principale.**

Nous avons pris la décision de conserver la fonction **setup** dans le fichier **main** car elle constitue en quelque sorte la **base de notre jeu** et est directement liée à la fonction principale, main. Cette approche nous permet de mieux structurer notre code, facilitant ainsi sa maintenance et son évolution.

Dans le fichier makefile il est important de rester vigilant et de ne pas substituer les espaces aux tabulations lors de la mise en forme du code. Cette précaution est essentielle pour maintenir la cohérence et la lisibilité du code source.

Voici un schéma qui montre comment les différents fichiers sont reliés dans le makefile.



LE FICHIER MAIN.C

C’est notre fichier source. Les appels de fonction se font dans ce fichier: dans une boucle général qui nous permet la création du jeu.

- Les variables sont initialement déclarées dans la fonction principale (main) afin d'éviter l'utilisation de variables globales. Pour garantir leur accessibilité dans les différentes fonctions, ces variables ont été passées en tant qu'arguments dans chaque fonction, et leur déclaration a été conservée dans le main.

- la fonction **setup** qui nous permet d’avoir le fond grâce au fonction de la bibliothèque graphique.

==> Les consignes sont “Le terrain de jeu prend la forme d'une grille de 40 lignes par 60 colonnes” le terrain représente ici le fond vert, l’arrière fond est le fond noir

```
InitialiserGraphique();
CreerFenetre(40,40, (TAILLEX+2)*LARGEUR, (TAILLEY+6)*HAUTEUR);
ChoisirCouleurDessin(CouleurParNom("black"));
RemplirRectangle(0, 0, (TAILLEX+2)*LARGEUR,  (TAILLEY+6)*HAUTEUR);
ChoisirCouleurDessin(CouleurParNom("green"));
RemplirRectangle(HAUTEUR, LARGEUR, TAILLEX*LARGEUR, TAILLEY*HAUTEUR);
```

Dans le main on a la boucle général. Celle-ci permet de:

-Faire appel aux fonction ainsi que leurs variables.

-Placer en argument pour toute les variables qui ont des pointeurs (*) on remplace les pointeur par des adresse.

Dans le **main.c** le programme entre dans une boucle principale qui continue tant que la variable **`gameOver`** est **fausse**. La boucle gère les entrées utilisateur, notamment les touches directionnelles, Escape et Espace.

- Si la touche Espace est pressée, le jeu peut être mis en **pause** ou **repris**, modifiant ainsi le comportement du jeu.

- Si le jeu n'est pas en pause, **le serpent** est **effacé**, **déplacé**, **redessiné** grâce aux appels de cette fonction, et le temps de la prochaine mise à jour est déterminé.

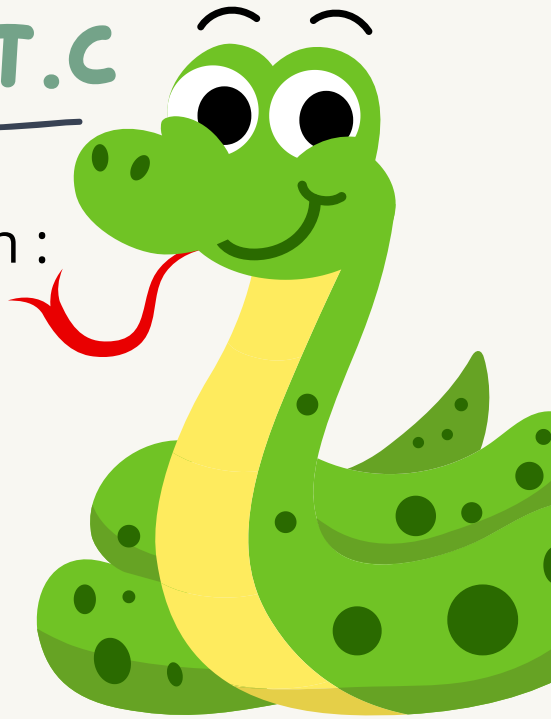
- Si le jeu est en pause, un rectangle blanc est affiché avec le texte "Jeu en pause" grâce à l'utilisation des fonction des bibliothèque graphique.

La fonction **`chrono`** est appelée pour mesurer le temps écoulé depuis le début du jeu, prenant en compte l'état du jeu (en pause ou en cours).

Enfin une attente est introduite pour réguler la vitesse du jeu (`attendre(CYCLE)`). La bibliothèque graphique est fermée à la fin de la boucle.

Nous avons également la fonction 'attendre()' qui introduit un délai en attendant que le temps actuel dépasse une valeur calculée en ajoutant la constante 'CYCLE' à l'heure actuel.Cela permet de créer des pauses ou des temporisations dans le programme.

GESTION_SERPENT.C



Dans ce fichier nous avons trois fonction :

- DeplacerSerpent
- DessinerSerpent
- EffacerSerpent

DeplacerSerpent:

Dans notre programme, nous utilisons la variable "**direction**" avec la valeur "**XK_Right**" pour permettre au serpent de se déplacer automatiquement. Pour modifier la direction du serpent, nous utilisons des conditions spécifiques. Ces conditions vérifient si la direction correspond à une touche spécifique. Si c'est le cas, nous actualisons la tête du serpent pour lui permettre de changer de direction.

Nous avons également , une condition dans cette fonction qui vérifie si le serpent touche les bords de la fenêtre noire , ce qui entraînerait sa mort. Si la nouvelle position de la tête du serpent dépasse la taille de la fenêtre, par exemple, le jeu se termine et le programme retourne un signal correspondant à "game over".

Au sein de notre fonction "DeplacerSerpent", nous intégrons également les vérifications de collision avec les pommes et les pommes pourries. Si les coordonnées de la nouvelle tête du serpent correspondent à celles d'une pomme, nous ajoutons deux segments à la variable "**snakelength**", représentant ainsi l'augmentation de la longueur, tandis que la variable de score s'accroît de **5 points**. La gestion des collisions avec les pommes pourries est expliquée dans la dernière section, étant l'objet de notre variantes.
La fonction **snprintf** nous permet de convertir le score en une chaîne de caractères dans la variable afficher_score. Enfin, nous utilisons les fonctions de la bibliothèque pour afficher le score à l'écran.

EffacerSerpent:
Cette fonction utilise les coordonnées du serpent en tant que paramètres. À l'aide d'une boucle, elle remplit chaque position du serpent avec un rectangle vert . Cette action a pour effet d'effacer graphiquement le serpent, ce qui crée l'illusion visuelle que le serpent "mange" une pomme, car à chaque mise à jour, le serpent est repeint en vert pour chaque nouvelle position.

DessinerSerpent:

Cette fonction prend en argument les coordonnées du serpent ainsi que sa longueur. Le paramètre '**j**' est utilisé comme compteur dans la boucle. Ensuite, nous sélectionnons la couleur noire pour représenter notre serpent en utilisant une fonction trouvée dans la bibliothèque graphique '**graph.h**'. La boucle sert à incrémenter le compteur 'j' de 1 jusqu'à atteindre la longueur du serpent. À chaque itération, une autre fonction de la bibliothèque graphique est utilisée pour ajouter un rectangle noir, permettant ainsi de dessiner le serpent jusqu'à obtenir la longueur désirée. Cette fonction nous permet donc de dessiner un rectangle de la longueur voulue, représentant ainsi visuellement notre serpent.

GESTION_CHRONO

Cette fonction commence par inclure trois bibliothèques : ‘**stdlib.h**’, `**stdio.h**` **,** et `**graph.h**`. De plus, il inclut un fichier d'en-tête "**gestion_chrono.h**", suggérant l'existence de fonctions ou de déclarations liées à la gestion du chronomètre dans un fichier séparé.

La fonction `**chrono**` prend deux paramètres en entrée : un pointeur vers une variable de type `**unsigned long**` nommée `**debutDuJeu**` et une variable de type `**int**` nommée `**jeuEnPause**`. .

Le code déclare deux variables statiques (`**dernierTemps**` et `**tempsEnPause**`) qui conservent leur valeur entre les appels successifs de la fonction. Elles sont utilisées pour stocker le dernier temps en microsecondes et le temps écoulé pendant les périodes de pause du jeu.

La variable `maintenant` est utilisée pour stocker le temps actuel en microsecondes, à l'aide d'une fonction externe appelée `Microsecondes()`.
e temps écoulé depuis le dernier appel à la fonction est calculé à l'aide de la différence entre le temps actuel et le `**dernierTemps**`.
La variable `**tempsTotal**` est utilisée pour stocker le temps total écoulé depuis le début du jeu, en prenant en compte le temps passé en pause.
Les variables `minutes` et `secondes` sont utilisées pour convertir le temps total en minutes et secondes.

Si le **jeu vient de commencer** (vérifié par `*debutDuJeu == 0`), le début du jeu est initialisé à l'instant actuel, et le `dernierTemps` est mis à jour. Si le jeu n'est pas en pause, le temps total est calculé, et si le temps écoulé depuis le dernier rafraîchissement graphique (`CYCLE`) est dépassé, le texte du temps est formaté et affiché à l'écran.

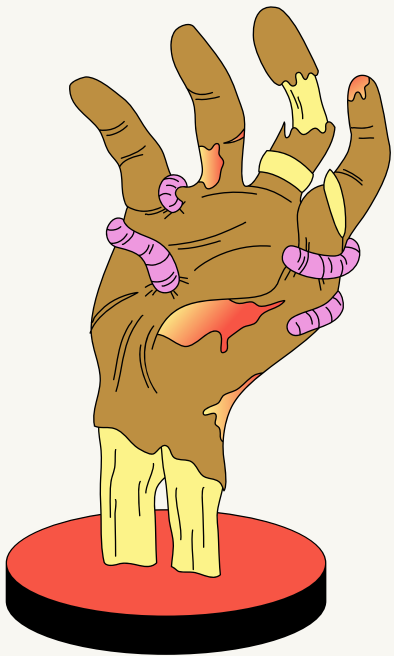
Si le **jeu est en pause**, le `**dernierTemps**` est ajusté pour éviter les erreurs d'affichage, la variable `**jeuEnPause**` est réinitialisée, et le `**dernierTemps**` est à nouveau réinitialisé pour éviter les écarts de temps après la pause.

/!\Choiisir la police=2

LA VARIENTE

Nous avons introduit une variante dans laquelle, si le serpent mange une **pomme pourrie**, il rétrécit jusqu'à atteindre sa **taille minimale**. Si le serpent atteint cette taille minimale, il meurt.

Afin de réaliser cela, nous avons introduit la fonction **srand(micros+1)** dans le fichier **main.c**. Cette fonction est utilisée pour **générer aléatoirement** l'apparition des pommes pourries. **L'ajout de "+1" a pour but d'éviter que les pommes pourries se retrouvent aux mêmes positions que les pommes classiques**. Ensuite, nous avons inclus une condition dans la fonction "**DeplacerSerpent**" : si les coordonnées de la nouvelle tête du serpent correspondent à celles d'une pomme pourrie, nous réduisons la longueur du serpent de deux segments. De plus, une autre condition a été ajoutée pour vérifier si la taille du serpent devient inférieure à sa taille minimale ; si c'est le cas, le jeu se termine avec un "game over".



Conclusion de Marwa:

Ce projet a été pour moi une expérience enrichissante car il m'a donné l'opportunité d'explorer de nouvelles connaissances, notamment en termes de résolution de problèmes rencontrés sur Git et de traitement des problèmes rencontrés dans les Makefiles ou plus généralement dans le code du jeu. Cela m'a motivé à faire des recherches approfondies pour mieux comprendre et mieux apprendre. J'ai particulièrement apprécié ce projet car j'ai ressenti la satisfaction de surmonter les obstacles et de faire progresser le jeu étape par étape jusqu'à ce qu'il soit enfin réalisé. Ce projet m'a également fait prendre conscience de mon vif intérêt pour le développement de jeux, malgré les défis auxquels j'ai dû faire face.

Conclusion de Djedjiga

Cette expérience m'a permis d'évoluer davantage en codage C89, qui était un langage jusqu'alors inconnu pour moi, et grâce à ce projet (et aux cour) j'ai pu avoir une approche intéressant. Le fait que ce projet combine les connaissances acquises en classe avec nos propres réflexions nous permet d'avoir de bonne connaissance.
De plus, nous avons rencontré d'autres problèmes en dehors du codage, notamment GIT, ce qui m'a vraiment permis de comprendre l'utilisation de ce dernier.
Cependant, je ne me vois pas travailler dans le développement de jeux vidéos plus tard.
En plus d'acquérir des compétences techniques, cette expérience m'a appris l'importance de la persévérance et de l'organisation.