

Rendu Ecrit

Castel Aurélien
Alex Maizeroi

Projet tutoré n°1

Sommaire

- I) Introduction
- II) Fenêtre et choix des difficultés
- III) Tableau de valeurs aléatoires
- IV) Partie graphique du memory
- V) Tableau de « flag »
- VI) Chronomètre et triche
- VII) Conclusions personnelles
- VIII) Dessins et cheminement de pensées

I) Introduction

Description du sujet

Le memory est un jeu qui se compose de cartes avec des illustrations.

L'ensemble des cartes est mélangé, puis étalé face cachée.

Le joueur retourne deux cartes de son choix. S'il découvre deux cartes identiques, elles restent retournées. Si elles sont différentes, le joueur doit les retourner à leur emplacement de départ.

Le jeu est terminé lorsque toutes les paires ont été retrouvées et retournées par paires.

Explication découpage programme et schémas

Notre memory est constitué de trois fichiers .c avec leurs librairies .h respectives. Pour comprendre cette répartition il faut regarder les fonctions qu'ils contiennent.

Le premier difficulty_tableau.c comprend le choix des difficultés. La fonction difficulty donne en retour la taille de la grille de jeu et la taille que devra avoir notre tableau aléatoire de paires, tout cela en fonction de la difficulté.

Dans difficulty_tableau.c on a aussi notre création du tableau aléatoire grâce à la fonction remplisseur_tableau.

Le fichier difficulty_tableau.c ne sera plus utilisé lors du lancement de la partie de memory c'est pour cela qu'il est séparé des autres .c, tous les autres .c seront utilisés lors de notre partie.

Maintenant graphismes.c contient toute la partie visuelle (images), interactive (sourisclic) et corrective (completeurflag).

On a deux tableaux différents : le tableau aléatoire et le tableau flag.

Le premier est utilisé pour les cartes : un numéro est associé à une image %d.png. C'est un tableau de valeurs.

Le second est un tableau de booléens et fait la même taille que le tableau aléatoire. Il permet de savoir si un utilisateur a cliqué (1) ou non (0) sur une case.

Les deux tableaux sont complémentaires comme si une case avec une certaine valeur avait aussi l'information 0 ou 1.

Chrono_triche.c contient les fonctions optionnelles, qui n'ont pas d'impact sur le fonctionnement du jeu mais qui fonctionnent toujours en même temps que les fonctions de graphismes.c .

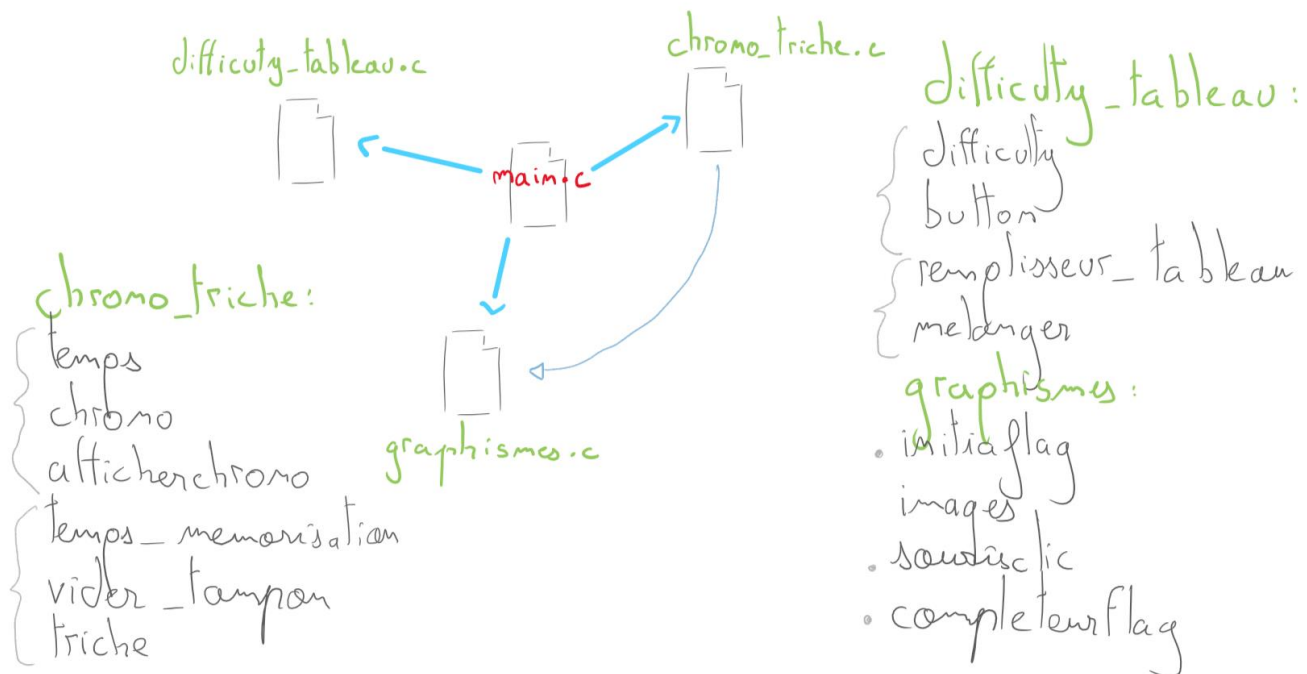


Schéma de dépendance des fichiers de notre memory et leurs fonctions respectives

Notre rapport s'organise comme l'ordre du schéma : par ordre chronologique du déroulement du programme.

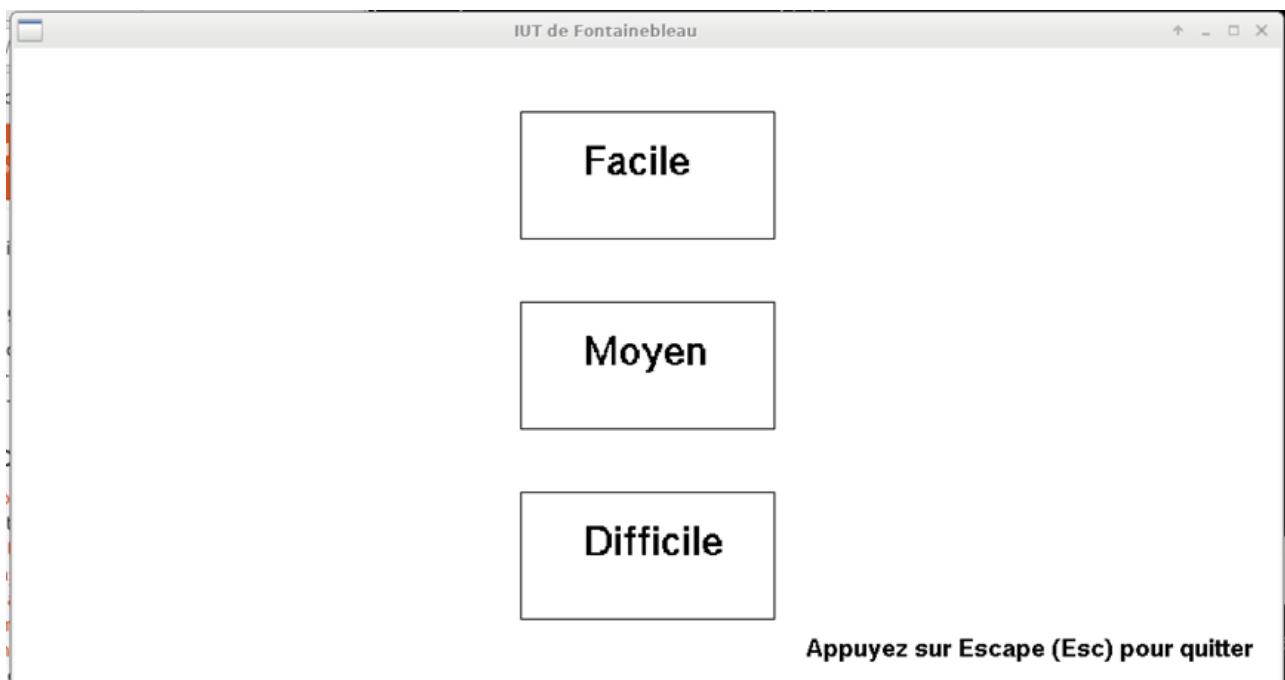
Ainsi nous expliquerons en commençant par expliquer difficulty_tableau.c puis graphismes.c et enfin chrono_triche.c et leurs différentes fonctions. Puis nous terminerons par nos conclusions.

II) Fenêtre et choix des difficultés

La première initiative a été de laisser le choix à l'utilisateur du nombre de lignes et de colonnes, mais après quelques jours de réflexion, on est passé à un système de niveaux qui était un peu plus simple à mettre en œuvre. Le jeu propose donc trois niveaux. Bien sûr, vous avez deviné : facile, moyen et difficile.

Le fichier `difficulty_tableau.c` contient la fonction `difficulty` permettant le choix des niveaux de difficulté. Il comprend également la fonction `button` interprétant les coordonnées en fonction des clics de la souris afin d'ouvrir une nouvelle fenêtre adaptée à la difficulté sélectionnée. Il ne se passe rien s'il ne clique pas sur l'un des boutons.

La fonction `difficulty` qui gère le menu renvoie un entier qui correspond à la taille du tableau.



Fenêtre difficultés

III) Tableau de valeurs aléatoires

Concernant le tableau de valeurs. Nous l'avons conçu au début du projet. Sachant pertinemment qu'on aurait besoin de quelque chose nous permettant de faire en sorte que deux images soient liées.

Pour cela, nous avons conçu un algorithme capable de remplir un tableau d'entiers qui possède deux fois une même valeur à la condition que le nombre d'éléments du tableau à remplir soit pair.

Ensuite, il nous fallait une autre fonction nous permettant de mélanger ce tableau de manière aléatoire en utilisant `srand`.

Je m'en suis donc servi pour mélanger nos valeurs pour que le `srand` tire uniquement une valeur représentant un indice du tableau qu'il doit mélanger.

À un moment, nous nous sommes dit que cela pouvait poser problème si l'utilisateur tirait deux fois le même indice, mais après l'avoir exécuté à la main la méthode a seulement quelques défauts. Plus le tableau est petit plus les chances qu'un nombre soit à côté de son égal dans le tableau est forte. Aussi, cela requiert un certain nombre d'itérations qui est assez important. Le tout est donc fait dans deux fonctions.

IV) Partie graphique du memory

Après avoir initialisé notre tableau aléatoire par paires, attaquons-nous à la partie graphique du memory.

La partie graphique du memory est à l'intérieur d'une nouvelle boucle « infinie », tant que l'utilisateur n'a pas fini de réaliser le memory.

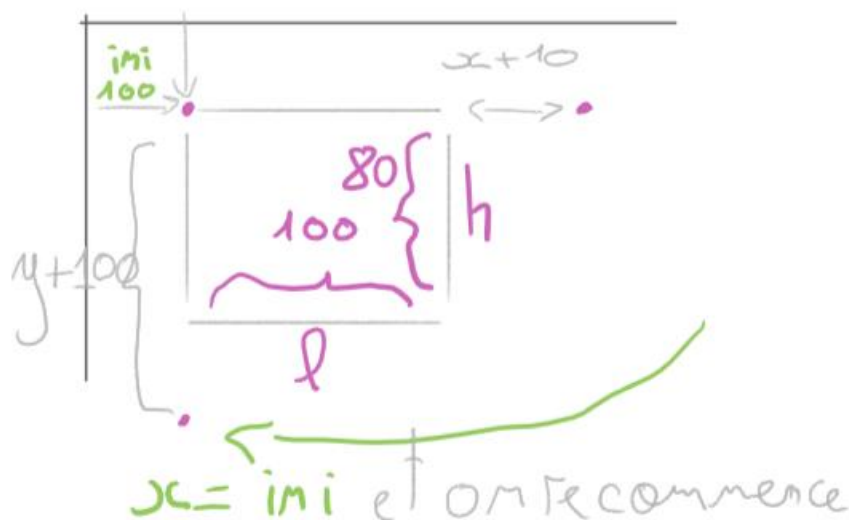
images(flag, tableau, longueurmax, hauteurmax);

La fonction images est constituée d'une boucle de deux for imbriqués, la principale pour le y et le secondaire pour les x. Ainsi nous avançons de x par pas de 100 jusqu'à arriver au bout et changer de ligne.

À l'intérieur de la boucle pour les x : on dessine des grilles bleues.

Et en fonction du tableau flag, on affiche si un zéro est présent : on met le dos de la carte sinon on met l'image de la carte correspondante avec le tableau aléatoire créé précédemment.

On utilise sprintf pour les images qui permet de mettre dans « buffer » le nom du répertoire et le nom de l'image en fonction de i (donc 0.png, 1.png... À chaque changement de x). Et on utilise ChargerImage en mettant buffer dedans.



V) Tableau de « flag »

Ce tableau est celui qui permet de compléter l'avancée de l'utilisateur. Quand l'utilisateur clique, ce tableau change.

`nombredeclics=sourisclic(nombredeclics,flag,longueurmax,hauteurmax);`

Cette fonction marche de la même manière que « images ». En utilisant `SourisCliquee` on met les valeurs des positions x et y cliquées dans `xclic` et `y clic`. Alors si l'utilisateur clique, on prend les coordonnées et on entre dans deux boucles imbriquées y et x. Si les coordonnées correspondent à une case du tableau graphique alors le tableau flag (en fonction des i) va avoir un 1 (s'il y avait un 0). On compte aussi le nombre de clics.

La fonction retourne à la fin `nombredeclics`.

`completeurflag(NB,flag,tableau,&jeu);`

Fonction qui sert à corriger le tableau flag. On le nettoie des paires de cartes non identiques que l'utilisateur a choisi, sinon si la paire a été juste on garde et on ne change pas.

La fonction est constituée d'une boucle principale pour les numéros de cartes. Et à l'intérieur une boucle qui correspond au tableau flag.

Ainsi on vérifie pour chaque numéro combien de fois ils ont été sélectionnés. Pour chaque case j on raisonne comme cela : si le numéro de la carte correspond à la valeur du tableau aléatoire et s'il y a un 1 alors on le compte et on sauvegarde cette position j.

Exemple : N°carte 2 : tableau [0] n'est pas un 2 on continue, tableau [1] n'est pas un 2 on continue, ... , tableau[4] est un 2 mais flag[4] n'a pas de 1 on continue, ... , tableau [26] est un 2 et a un 1 on compte compteur+1 et on sauvegarde 26 dans `sauvgardeurj`.

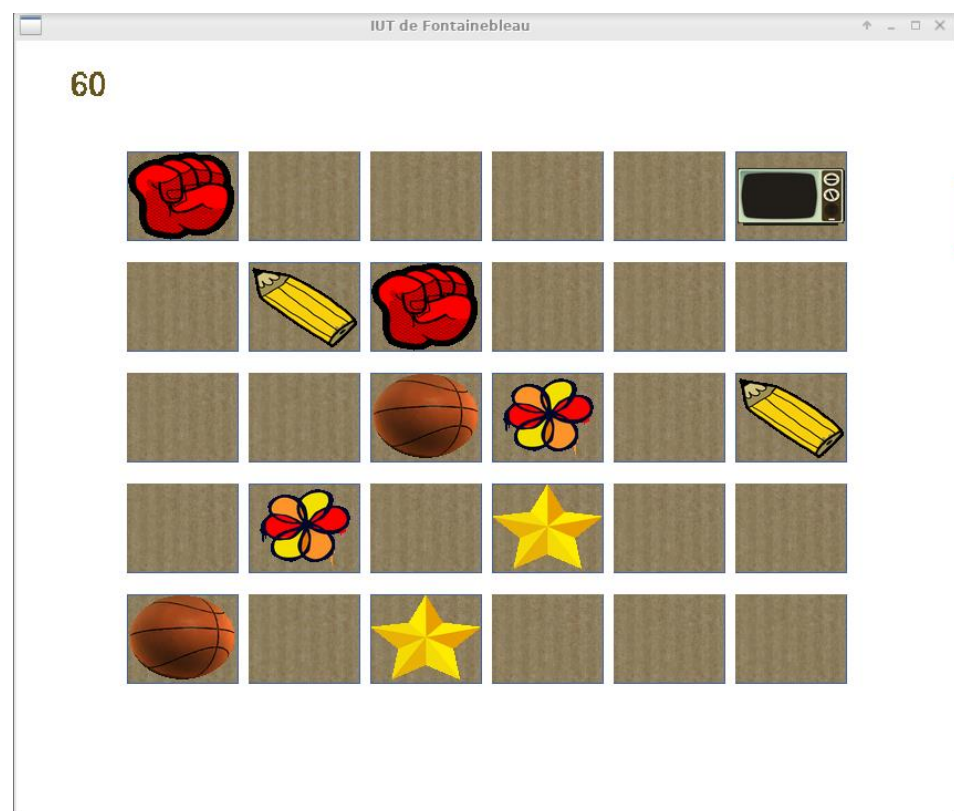
Après avoir terminé la vérification pour un certain numéro de carte on finit la boucle et on va corriger si le compteur est égal à 1. Et on compte combien il y a eu d'erreurs globalement avec `compteurdepasbon`.

Exemple : N°carte 2 : le compteur est égal à 1 , pour flag[26] qui était égal à 1 on le remet à 0 et on incrémente `compteurdepasbon`.

Lorsqu'on a terminé de vérifier toutes les cartes on vérifie compteurdepasbon.
S'il n'y a eu aucune erreur lors de la vérification alors le memory est terminé et
jeu sera égal à 1.



Début de la partie : tout est retourné, tous les flags sont égaux à 0



Partie qui se complète : certaines cases de tableau flag se complètent au fur et à mesure

VI) Chronomètre et triche

Chrono : La fonction temps récupère le temps et chrono va faire la soustraction entre deux temps.

```
enregistreur=tmpr; /*Chrono, réitération*/  
tmpr=chrono(tmprini,tmprmax);  
if (enregistreur!=tmpr)  
{ ...
```

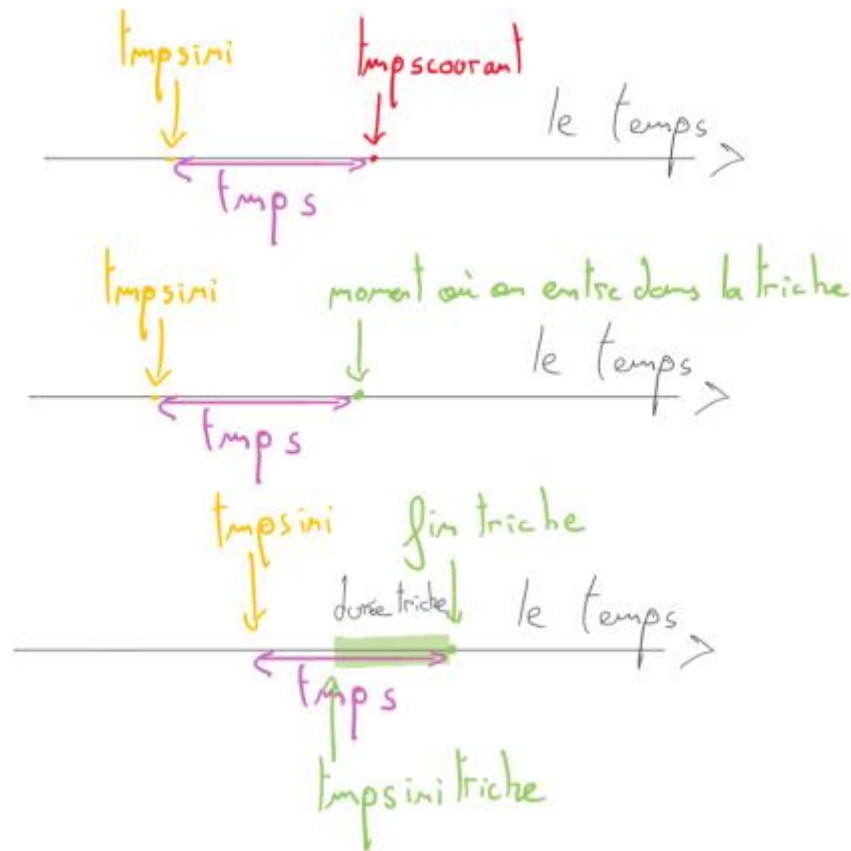
On sauvegarde l'ancien temps dans la variable enregistreur et on recalcule un nouveau tmpr après. La condition if sert à afficher uniquement les secondes différentes par rapport aux précédentes grâce à enregistreur.

Dans la condition on utilise aussi une surface de blanc à la place des anciennes secondes pour les cacher.

Triche : On réutilise la même méthode générale, mais au lieu d'utiliser le tableau flag la fonction triche génère un autre tableau flag mais rempli uniquement de 1 comme ça toutes les images sont retournées. L'utilisateur est bloqué dans une boucle tant qu'il n'a pas réappuyé sur la touche t.

Toutes les interactions utilisateur (clique et clavier) sont nettoyées grâce à la fonction vider tampon. On nettoie les tableaux qui contiennent les positions souris et les touches clavier dans une boucle en appelant SourisCliquée et Touche « dans le vide ».

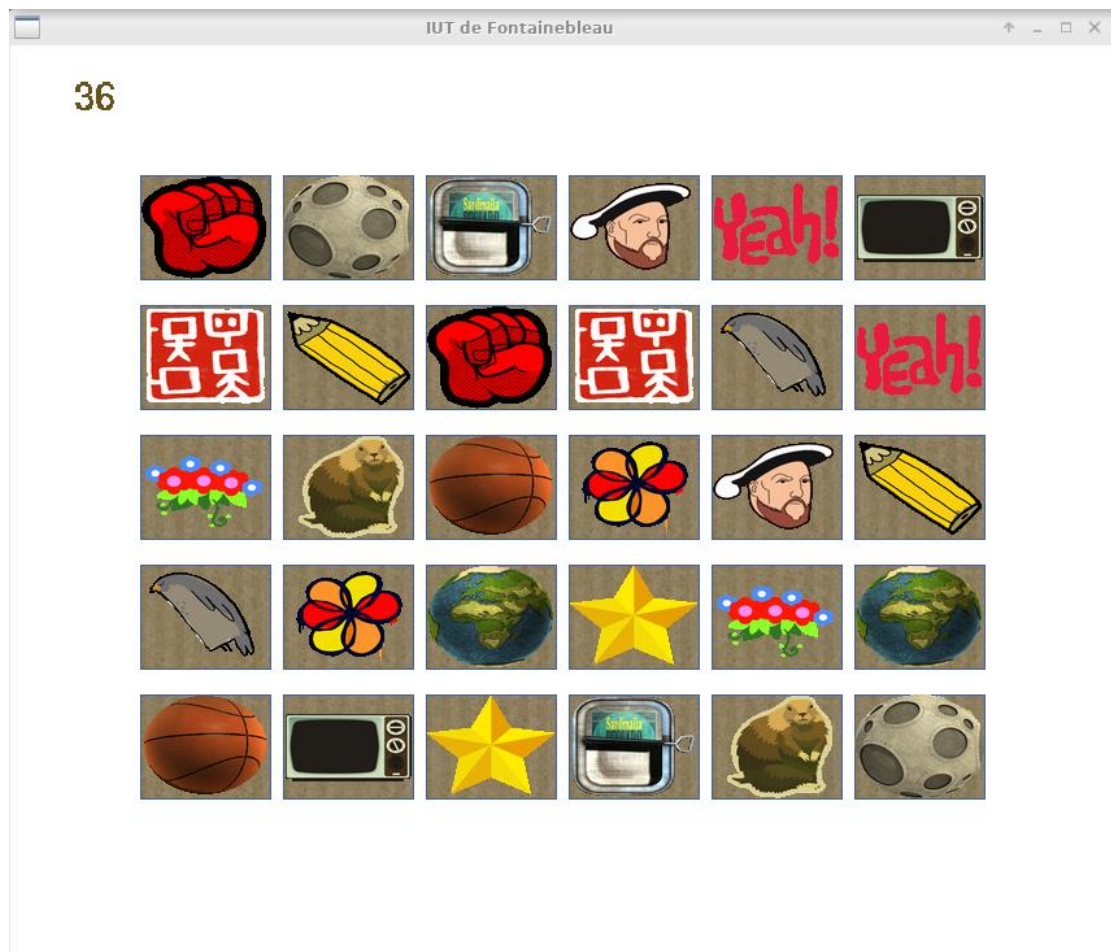
Voici comment le temps reste inchangé après avoir utilisé la triche :



Temps mémorisation : est utilisé seulement après avoir fait 2 clics et s'il y a eu des erreurs de l'utilisateur dans la fonction completeurflag. De cette manière les utilisateurs qui ne font pas d'erreurs peuvent rapidement finir la partie. On utilise la fonction temps_memorisation qui est légèrement différente de la fonction temps. Comme temps est trop imprécise pour compter une seconde on utilise Microsecondes.

À l'intérieur du do while (dans le main.c) on réutilise de la même manière l'enregistreur pour continuer à afficher le chronomètre qui défile.

Comme l'utilisateur ne peut rien faire durant cette période on utilise la fonction vider tampon.



Mode triche

VII) Conclusion personnelle Aurélien

Je me suis occupé des parties suivantes graphique.c, chrono_triche.c et les appels de leurs fonctions dans le main.c.

Au début, ce n'était pas évident de savoir par quoi se lancer entre la partie graphique et traitement des valeurs. Je me suis d'abord occupé de faire le chronomètre parce qu'on avait fait les structures peu de temps avant et que cette partie n'était pas dépendante du reste du memory.

C'est à ce moment-là que je me suis mis avec Alex, car nous travaillions déjà ensemble lors des TP et nous voulions gagner plus de temps en mettant en commun nos travaux.

Lui avait déjà réalisé la partie tableau aléatoire alors je me suis mis à faire la partie graphique. Cela n'était pas évident au début de créer un processus de création automatique de la partie visuelle du jeu. Mais après m'être approprié la bibliothèque et fait des boucles imbriquées (y, x) tout s'est ensuite fait naturellement, ma grille de jeu et mes cartes.

Le seul problème était que la bibliothèque graphique ne marchait pas chez moi alors il fallait que je sois très précautionneux pour ne pas recommencer à zéro tout mon travail du week-ends (mais au moins ça m'a entraîné à marcher plus sur de la théorie plutôt que de l'expérimental).

Il me manquait l'interaction utilisateur et l'avancée utilisateur (tableau de flag), je me suis réuni avec Alex pour savoir comment faire marcher SourisCliquée et tout s'est ensuite fait naturellement.

Après avoir fini tout ça, nous avons réunis nos codes pour faire le makefile, ce fut d'une grande satisfaction de pouvoir enfin avoir quelque chose de proche du final et qui marchait bien (même s'il manquait encore des choses).

Pour le reste, comme la partie triche nous nous sommes mis ensemble pour le faire.

À la toute fin du projet nous nous sommes mis à la correction de bug. Avoir un binôme a été d'une grande aide pour avoir un œil différent et aussi car il avait des parties du programme différentes des miennes.

Nous avons donc fait beaucoup de tests différents jusqu'à par exemple corriger un bug qui permettait d'accéder au mode triche après le temps de mémorisation.

À deux nous nous complétions bien à corriger notre projet et je pense que j'aurai laissé passé beaucoup de bug en travaillant seul.

Le projet dans son ensemble n'était pas évident à commencer. Il fallait structurer le programme en fonctions et les appeler depuis un main.c en veillant au passage des paramètres. Je suis content du résultat obtenu pour le code dans son ensemble, car je réutilise les fonctions que j'ai précédemment créées (comme triche ou chrono). Au début, je n'avais pas forcément pensé à construire mon code comme cela, mais finalement ça m'a été bien utile. Je me suis amélioré dans les boucles, car créer une grille de jeu graphiquement était totalement nouveau. J'ai aussi beaucoup utilisé les tableaux (l'un pour les graphismes et l'autre booléen). J'ai utilisé aussi des pointeurs pour faire des passages par adresse au niveau des argument des fonctions pour que les variables soient réellement modifiées après l'appel de la fonction.

Ce qui pourrait améliorer notre memory au-delà du projet :

L'affichage est sommaire, ce n'était pas notre but de faire quelque chose de joli mais je pense qu'on pourrait le rendre plus agréable à jouer.

Faire des animations pour le rendre plus interactif et pour que le temps de mémorisation soit moins saccadé visuellement.

Pour faciliter le jeu pour l'utilisateur il aurait été intéressant d'introduire les règles du jeu dans une fenêtre. Comme ça n'était pas demandé on n'en n'a pas fait mais cela aurait été utile si le programme avait été utilisé par un utilisateur extérieur à l'IUT.

VII) Conclusion personnelle Alex

Pour comprendre comment je l'ai vécu, je trouve important de notifier qu'à ce moment de la formation, je n'avais pas une forme olympique. Je dirais même que ça allait mal. Au début, j'étais dans une panique assez immense voyant le projet que l'on m'avait demandé de faire insurmontable. Autant dire le devoir qui signait la fin de ma carrière avant même qu'elle ait commencé. Mais puisque je n'ai pas pour habitude d'attendre mon échec venir sans me battre, je me suis attaqué à la partie où je trouvais que je me débrouillais le mieux. Donc j'ai commencé à faire tout ce qui était algorithmique. Très vite, j'ai commencé à échanger avec Aurélien afin de s'entraider jusqu'au moment où on a décidé de travailler de manière officielle en groupe rendant les choses beaucoup plus simples et ce qui a permis de faire avancer le projet d'un pas-de-géant. Malgré ma faible maîtrise de la bibliothèque graphique. Aurélien était un peu plus à l'aise et vu que j'étais plus à l'aise sur l'algorithmie. Finalement, on était complémentaire. Ensuite, lorsqu'il a fallu patcher les bugs le fait d'être à deux a été bénéfique. Je pense que le fait que l'un code et que l'autre regarde a pour effet d'apporter un regard clair qui permet de mieux trouver les bugs et de s'expliquer l'un l'autre pourquoi ça ne marchait pas.

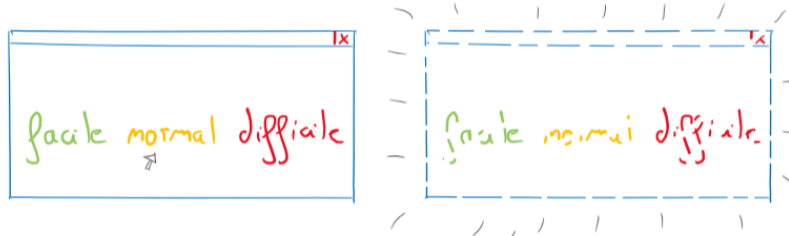
Pour finir, sachant qu'à ce moment-là, je n'étais pas sûr de ma présence au sein de la formation au vu de mes compétences et de mes résultats. Je me suis rendu compte que j'étais capable de faire des choses intéressantes contrairement à ce que je pensais d'autant plus que je m'étais rendu compte plus tard que le dernier échec au contrôle d'APL était dû à une mauvaise compréhension du sujet. C'est pourquoi à l'avenir au lieu de rester bloquer dans mon coin. En fin de compte, il n'y a pas de honte à demander de l'aide, je dirais même que c'est recommandé.

J'améliorerai bien le graphique histoire que ce soit un peu plus agréable à l'œil. Après aussi faire un système de défi avec des médailles d'or d'argent et bronze. Et pourquoi pas aussi faire un top 10.

VIII) Dessins et cheminement de pensées

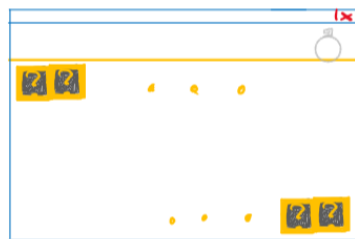
Divers dessins qui nous ont aidé pour créer notre memory :

1: difficultés



2: le tableau se crée en fonction de la difficulté (taille différente)

3: graphismes



les ? sont des caches, derrière ils y a les vraies images

pour faire disparaître les caches on met leur opacité à 0%

Si l'utilisateur clique sur une case elle sera découverte
 L> si la prochaine image est différente la seconde image est découverte et les deux images sont recouvertes 1 sec après



est découverte et les deux images sont reconstruites 1 sec après
 ↳ si la prochaine image est la mère: la seconde image est découverte, les deux images sont découvertes pour toujours




ce qui il ya à la fin

boucle :

L est à calculer, il est nécessaire d'avoir une itération différente car le tableau n'a pas la même taille en fonction de la difficulté

on passe d'une même ligne à une autre



x et y initialiser \rightarrow $x = ?$
 boucle \rightarrow $y = ?$
 On prend comme point de structure du tableau
 boucle $x \rightarrow$ $\text{for} (y \leq \text{hauteur max}; y = y + L; i++) \{$
 $\text{for} (x \leq \text{largeur max}; x = x + L; i++) \{$
 mettre image n° en fonction du tableau

mettre image n^o en fonction du tableau
tableau[i], ...
créer cache (x, y); }

Remerciements aux professeurs Luc Hernandez et Denis Monnerat pour leur aide