

# LE MEMORY

Anne-Sophie Besnard et Gaëtan Loddé

2018



## Table des matières

Introduction.....	2
Description des fonctionnalités du programme .....	2
Le menu .....	2
Le timer.....	3
Les différents modes de difficultés.....	3
Le mode triche.....	4
Les fonctionnalités bonus.....	5
L'affichage des règles du jeu.....	6
Le bouton retour.....	6
Présentation de la structure du programme .....	7
L'utilisation de fonctions .....	7
L'utilisation de Makefile .....	8
Exposition de l'algorithme qui remplit la grille de jeu .....	9
Le placement aléatoire des cartes .....	9
Explication de la forme des données qui représentent la grille de jeu .....	10
Conclusion .....	11
Conclusion d'Anne-Sophie Besnard.....	11

## Introduction

Le projet demandé à la fin du semestre 1 de la première année du DUT est de réaliser un Memory. Le joueur doit retrouver chaque paire de carte une à une. Le jeu s'arrête quand le joueur a trouvé toutes les paires de cartes.

Afin de réaliser ce projet, nous devons prendre en compte le temps de jeu, le temps qui s'arrête lorsqu'on retourne des cartes. En effet à ce moment-là, l'utilisateur ne peut faire aucune action sur la grille de jeu. Ainsi que le fait que le joueur ne peut retourner que deux cartes à la fois.

Afin de mieux comprendre notre programme, une description des fonctionnalités permettra de décrire la création du menu principal, du timer, des modes de difficulté et du mode triche. Quelques fonctionnalité bonus ont été mises en œuvre et seront expliquée.

La structure du programme se fait par l'utilisation de fonction et de Makefile afin de permettre une lecture plus simple du programme.

Enfin l'algorithme réalisé permettant de remplir la grille de jeu est exposé ainsi que les données utilisées pour faire la mémorisent. C'est-à-dire, de comparer les sprites entre eux par exemple.

## Description des fonctionnalités du programme

### Le menu



**Menu principal**

Afin de lancer le jeu, une interface graphique a besoin d'un menu avec différents boutons. Cela a été le point de départ dans la réalisation du projet.

Il a fallu initialiser le graphique puis créer la fenêtre, pour ensuite ajouter le fond de notre choix, créé sur Photoshop. Afin de trouver facilement les coordonnées de nos boutons pour ensuite ajouter la fonction de clique, on a créé des rectangles/carrés correspondant à nos boutons, cependant ces derniers seront effacés du programme ensuite, ils permettent juste de positionner les boutons. Il existe sur la fenêtre des menus quatre boutons :

- Un pour chaque difficulté : facile, normal et difficile.
- Un bouton qui affiche une autre fenêtre avec les règles du jeu contenant un bouton retour vers le menu principal.

On a créé une fonction Menu () qui regroupe toute la gestion de la fenêtre menu ainsi que les boutons qui renvoient aux différentes grilles de jeu suivant le mode de difficulté. Il comporte donc les fonctions Facile (), Normal () et Difficile () qui gèrent les grilles.

Afin de déterminer la zone cliquable des boutons, il a fallu récupérer la position de la souris lorsqu'elle clique avec la fonction SourisCliquee () puis en fonction du x, y, longueur et largeur des rectangles contenant les positions des boutons, calculer la surface cliquable. L'utilisation d'un if est nécessaire afin de savoir quelle zone renvoie à quelle fonction parmi la grille difficile, normal, facile ou règle du jeu.

Les fonctions des 3 modes de difficultés se basent sur le même principe ; elles affichent la grille dont le nombre de cases change en fonction de chaque difficulté, plus la difficulté est élevée plus il y a de cartes.

## Le timer

Le timer calcule en microsecondes le temps qui s'écoule dès que le joueur a cliqué sur l'un des boutons de difficulté, soit quand une partie commence. Il s'affiche en haut à gauche de la page. Toutefois, il a fallu le convertir en seconde afin de l'afficher correctement et de rendre la lecture du temps plus compréhensible.

La fonction EcrireTexte () a été utilisée afin de rendre visuel le timer. Il a fallu dans un second temps gérer le rafraîchissement de la page. En effet sans cela, les chiffres se superposent et cela devient illisible. Un rafraîchissement de la page a permis d'éviter cette superposition des secondes. Le timer a donc été réalisé dans un des écrans virtuels à notre disposition, puis une fonction CopierZone utilisée dans une boucle durant la partie permet le rafraîchissement du timer, et par conséquent éviter la superposition des chiffres.

On a initialisé la variable seconde à 0 afin que le timer parte de 0. Ensuite on initialise temps qui est égal à la fonction Microseconde () + Cycle qu'on définit plus haut. L'utilisation d'un if permettra de traduire que si Microseconde () (à définir) est inférieur à temps alors le temps s'incrémentera.

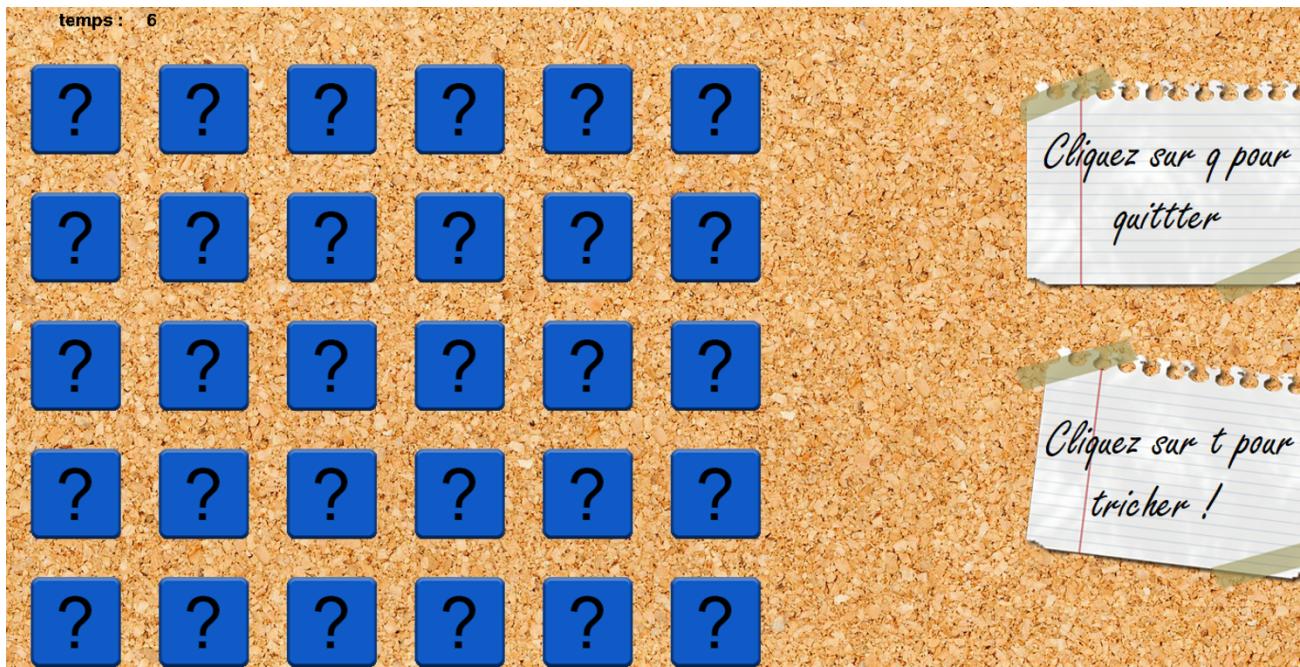
Un tableau de char, appelé buf[100] permet de contenir les secondes qui s'écoulent.

## Les différents modes de difficultés

Les différents modes de difficultés se traduisent dans le menu par plusieurs boutons. C'est alors à l'utilisateur de choisir quel mode de jeu il souhaite avoir. Il en existe 3 :

- Facile, où la grille est définie en 4x5 cartes
- Normal avec une grille de 5x6 cartes
- Difficile avec une grille de 5x8 carte

La touche q permet de quitter ses modes de jeu et de revenir au menu principal.



**Exemple du jeu sur la grille normale**

### Le mode triche

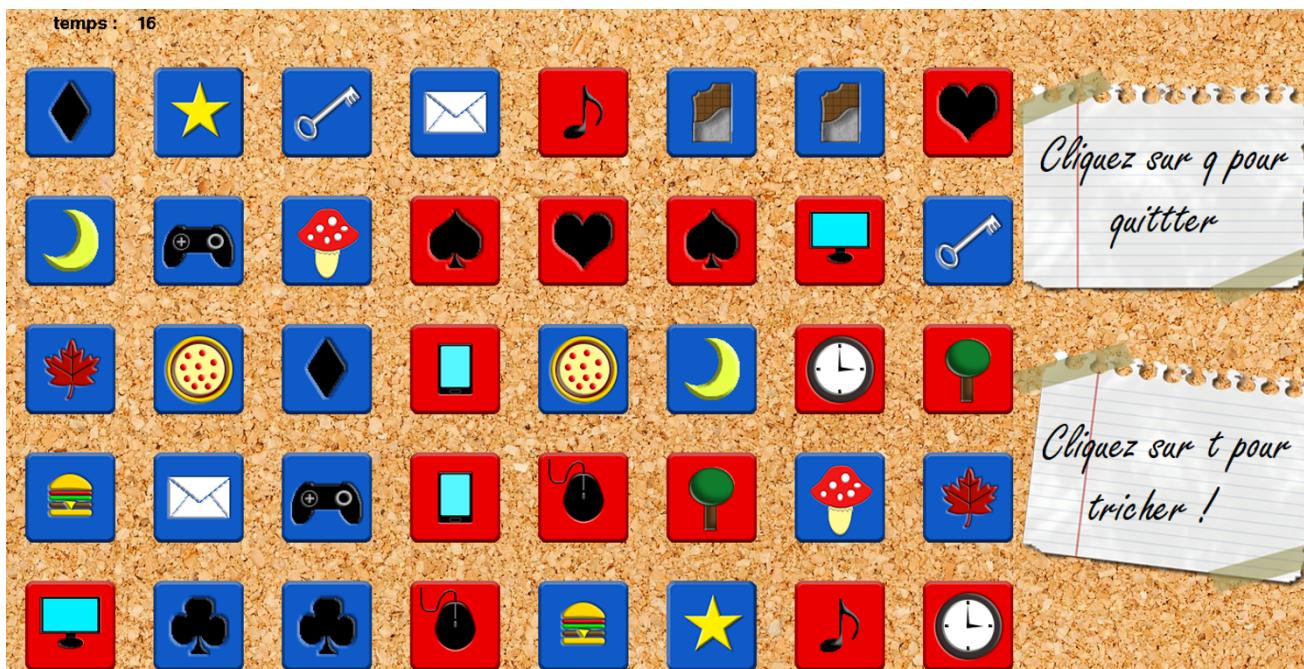
En appuyant sur la touche « t » une première fois, le joueur active le mode triche, ce qui lui permet de voir où se trouve toutes les cartes, elles sont donc découvertes. En réappuyant sur le bouton, cela doit disparaître et le joueur peut continuer sa partie là où il en était resté, les cartes déjà découvertes sont donc sauvegardées.

Lorsqu'on appuie sur la touche t le timer s'arrête et met en pause le jeu. La reprise du jeu reprend si l'on réappuie sur t.

Afin d'intégrer un cheatmode dans le programme, l'utilisation de trois CopierZone est nécessaire :

- Le premier sert à récupérer sur un écran virtuel (dit tampon) les cartes découvertes sur l'écran actif au moment de l'appui sur la touche et par conséquent de garder une sauvegarde des cartes retournées.
- Le second permet de récupérer sur un autre écran virtuel l'affichage de la disposition des cartes que nous aurons préalablement initialisées
- le troisième CopierZone qui récupère l'écran tampon et permet de recharger les cartes que le joueur a déjà retournées.

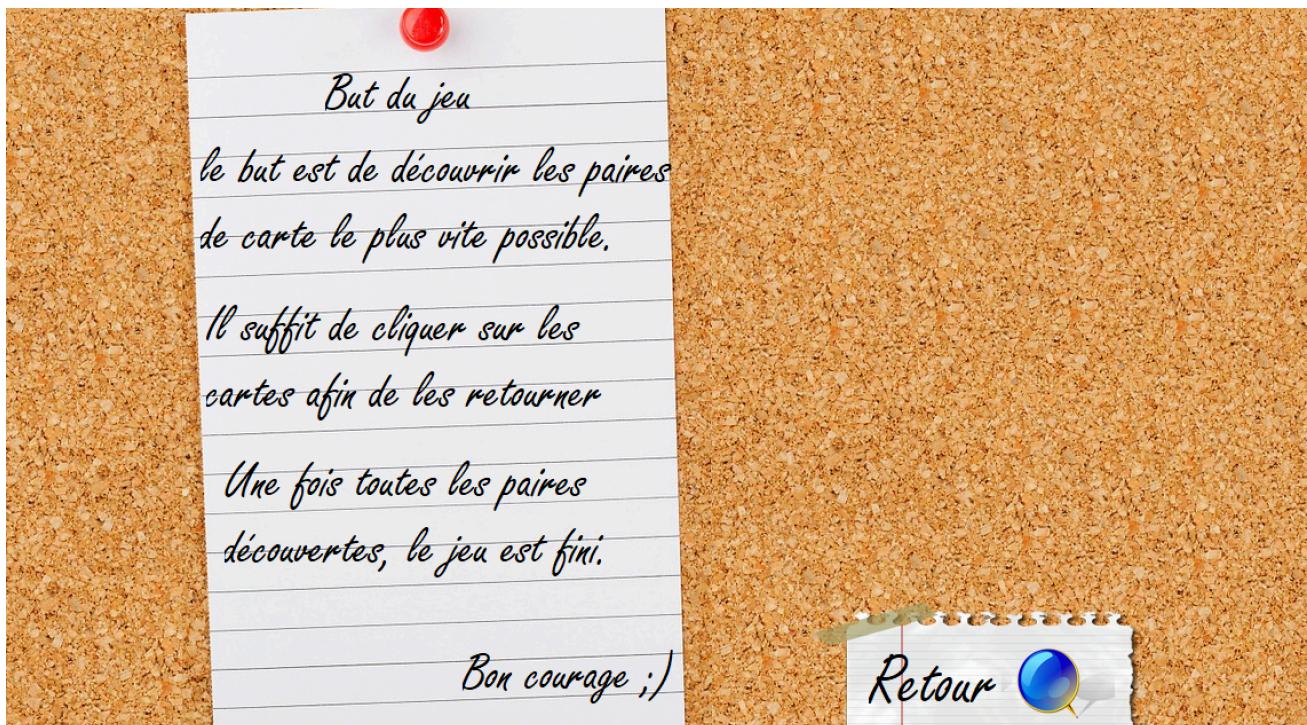
Préalablement nous avons chargé sur les 2 écrans virtuels les mêmes fond d'image, afin que lors de la superposition des copies d'écrans, le fond ne change pas et n'empieète pas sur le timer.



Un exemple du cheatmode sur la grille difficile

Les fonctionnalités bonus

## L'affichage des règles du jeu



## Ecran des règles du jeu

Dans le menu, le joueur peut noter la présence d'un bouton règle du jeu. S'il clique dessus, le fond du menu sera remplacé par le fond qui affiche les règles du jeu. Il suffit à l'utilisateur de cliquer sur le bouton retour afin de revenir au menu de sélection des niveaux.

## Le bouton retour et écran de victoire

À différents moments du jeu, le joueur peut remarquer la présence de boutons retour afin de revenir au menu, nous avons effectivement pensé que le joueur devait pouvoir revenir au menu quand il le souhaite et non d'attendre la fin de la partie pour y retourner.

A la fin du jeu, lorsque toutes les cartes ont été trouvée, un écran de victoire s'affiche et un bouton permet de revenir au menu principal.



## Présentation de la structure du programme

### L'utilisation de fonctions

L'utilisation de fonctions a été nécessaire surtout dans le fichier principal que constitue le main.c, cela permet de rendre plus clair et plus modulable le programme. En effet, l'utilisation de la fonction Grille(int cartex, int cartey), peut se faire autant de fois qu'il y a de difficulté. C'est à dire, que dans le programme main.c on l'utilise 3fois ;

- Pour le mode facile avec comme paramètre Grille(4,5)
- Normal avec Grille(5,6)
- Difficile avec Grille(5,8)

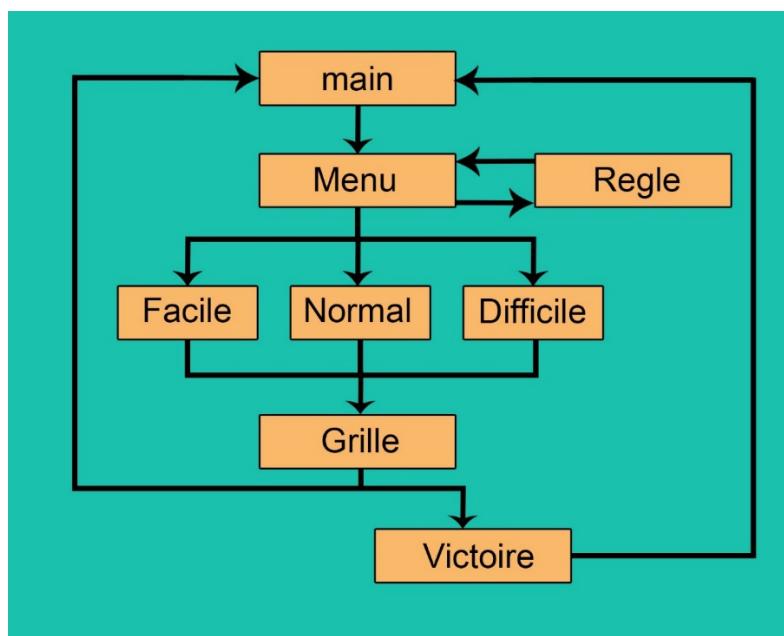


Schéma des appels de fonction

## L'utilisation de Makefile

Pour rendre plus lisible le main.c, ce dernier a été séparé en 2 fichiers : main.c et grille.c. grille.c contient la fonction Grille(int cartex, int cartey) qui permet de charger les sprites, d'effectuer la mise en place aléatoire des sprites, de gérer le timer, le clique ainsi que le cheatmode.

Cela a pour but de rendre plus lisible les fichiers en limitant leurs lignes de code à environ 150 lignes.

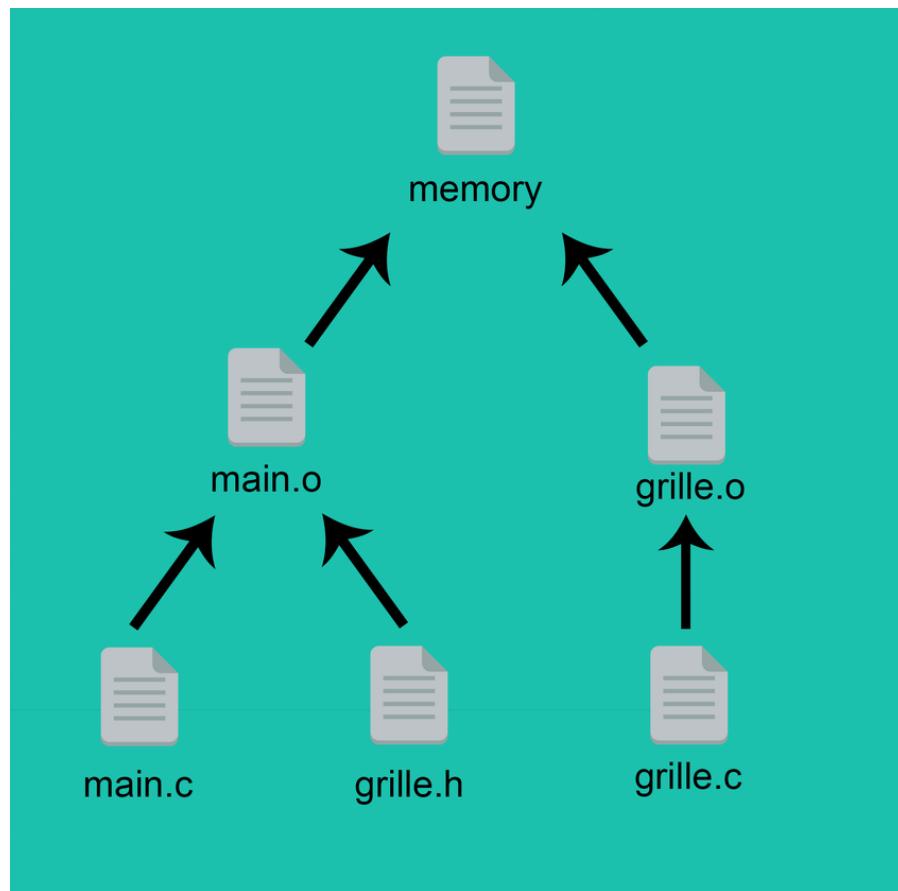


Schéma du Makefile

## Exposition de l'algorithme qui remplit la grille de jeu

### Le placement aléatoire des cartes

Pour effectuer le placement aléatoire des cartes dans la grille de jeu, il a fallu tout d'abord initialiser un tableau contenant tous les sprites et un tableau destiné à recevoir les cartes mélangées du premier tableau avec 2exemplaires de chaque sprite maximum. Le tableau contenant l'aleatoire (`spriterand`), a été initialisé comme contenant que des 0. À noter que le tableau `spriterand` a une capacité 2\* supérieure à celle du tableau de sprite, car elle contient 2exemplaires de chaque sprite.

Une première boucle parcourt le tableau de sprite afin de garder avec `current` chaque sprite et de la placer aléatoirement dans le tableau en 2 exemplaires. Pour le placer aléatoirement, l'utilisation de `rand` a été nécessaire pour attribuer une place aléatoire dans le tableau `spriterand`, si `spriterand[i] == 0` alors le sprite contenu dans `current` peut être placé, car 0 ici signifie que le tableau est « vide » de sprite. La boucle se fait une seconde fois afin de placer sa paire, puis la toute première boucle parcourt de nouveau le tableau de sprite pour contenir une autre valeur...

Une fois tous les sprites placés en paires dans le `spriterand` fini, on peut créer la grille de jeu et afficher les cartes. Il a fallu pour cela créer un tableau en 2dimensions et l'incrémenter. On désigne ensuite le tableau à

2 dimensions égal au tableau spriteRand créé précédemment, qu'on incrémente avec i++ pour parcourir les valeurs de spriteRand et de les afficher dans une grille.

Cependant on n'utilise pas tout de suite la fonction AfficherSprite() avec les valeurs contenues dans le tableau, car cela reviendrait à afficher où est quelle carte, elles ne sont donc pas cachées ! à la place on affiche directement le sprite destiné à être le dos de carte.

C'est le clic de la souris qui fera apparaître le sprite du tableau de sprite sur le sprite du dos de carte.

Afin d'avoir les coordonnées où les sprites se trouvent et où l'utilisateur peut cliquer, on a utilisé une boucle for afin d'incrémenter les zones cliquables. L'utilisation de SourisCliquee() est nécessaire afin d'obtenir les coordonnées cliquables et de savoir si l'utilisateur clique bien au bon endroit.

Dans la boucle se trouve la fonction if qui contient l'aire cliquable. Si l'utilisateur clique alors la fonction AfficheSprite() ; permet d'afficher le « véritable » sprite à cet endroit précis. Un i++ a dû être utilisé après le if pour incrémenter le tableau spriterand[i] et afficher le bon sprite qui se trouve à i endroit et d'avoir la bonne incrémantation. Avec le clic sur le sprite affiché par-dessus le dos de carte. La carte est alors « retournée ».

## Explication de la forme des données qui représentent la grille de jeu

La grille de jeu étant créée avec les fonctions décrites dans la section supérieure, le jeu en lui-même doit être fait. C'est-à-dire que les int créé à partir de ChargerSprite() doivent être utilisés afin de les comparer entre eux, de savoir s'ils forment une paire ou non.

Afin que les cartes déjà trouvées ne soient plus utilisables avec leur int, un autre tableau a dû être initialisé afin de stocker ses valeurs (spritetrouve[k]) ainsi qu'un booléen dejatrouve = 0 si faux et 1 si vrai.

Cela se trouve dans le fichier grille.c, plusieurs variables entre en jeu tel que selection, selectionx et selectiony. Ainsi si l'on clique sur une zone cliquable, si selection == 0 et que dejatrouve = 0, alors la variable selection va prendre la valeur du Spriterand[i] avec selectionx = x et selectiony = y pour ses coordonnées précises.

Pour éviter que le joueur clique 2fois sur la même carte, la gestion du second clique comprend plusieurs conditions :

- La variable selection (qui contient spriteRand[i]) doit être égale à la seconde carte, donc à spriteRand[i]
- Les x ou les y du second clique doivent être différents. On utilise || et non && sinon les sprites identiques contenus sur la même colonne ou même ligne seront considérées comme fausses.

Une autre condition s'ajoute ensuite, en effet il faut ajouter la valeur de selection dans le tableau spritetrouve[k] afin de rendre insélectionnables les cartes précédemment trouvées, cela empêche le joueur de tricher.

Si le second clique ne corresponds pas au premier, il est alors considéré comme faux et le backcard se réaffiche pour cacher les cartes.

À chaque fois qu'une carte est trouvée, la variable score s'incrémenter jusqu'à atteindre spritesize qui correspond aux nombres de paires mis dans la grille de jeu.

## Conclusion

### Conclusion d'Anne-Sophie Besnard

Le projet a été intéressant à réaliser, cela permet de mettre en œuvre toutes nos connaissances acquises lors du semestre 1.

Cependant, je pense que l'utilisation de struct pour les cartes, aurait été plus judicieux mais l'utilisation de pointeur et de struct en lui-même reste assez difficile pour moi encore. Cela aurait permis de réduire les lags présents dans le jeu, notamment avec le cheatmode quand on appuie sur <t>, il y a un grand temps de latence.

Également si je maitrisais plus l'utilisation de pointeurs j'aurais pu diviser encore plus le fichier grille.c pour le rendre plus lisible et réduire le nombre de lignes tout en rajoutant seulement un fichier .c en plus. Il y aurait eu un fichier .c contenant seulement la création de la grille et un autre contenant le jeu en lui-même, c'est-à-dire avec les gestions de clique ou des pairs par exemple. Dans ce cas-là, le Makefile aurait dû être aussi modifié.

Je pense aussi que j'ai mal organisé mon temps, j'ai bloqué pendant trop de temps pour comprendre et utiliser struct pour finalement laisser tomber l'idée. Cela a pu nous pénaliser au niveau du délai et ainsi limiter notre temps pour faire les bonus et optimiser le jeu avec des CopieZone().

Nous avions eu l'idée de faire un ladderboard ainsi qu'un compteur de score (qui afficherait les cartes restantes) comme bonus mais nous n'avons pas eu le temps pour.

Mais je pense qu'on fait un projet correct, peut-être pas assez optimisé ou peut être que notre programme a encore trop de lignes pour être correctement lu, mais l'ajout de nombreux commentaires peut aider à la compréhension.