

RAPPORT PROJET INFORMATIQUE :

LE MEMORY



Par : Aurélien Peden
Marie Pellier

SOMMAIRE

Introduction.....	3
Conception.....	4
Structure.....	4-5
Fonctionnalités.....	6-13
Pour aller plus loin.....	14-16
Conclusions.....	16

INTRODUCTION

Le projet “Memory” était pour nous le premier projet de notre DUT informatique. Ce projet avait pour but non seulement, de nous familiariser avec le travail de groupe en informatique, mais aussi de nous donner la possibilité de faire un travail sur la durée.

Pour ce faire, on devait utiliser la bibliothèque graphique de l’iut qui utilise XLIB et qui contient toutes les fonctions nécessaires à la réalisation du projet.

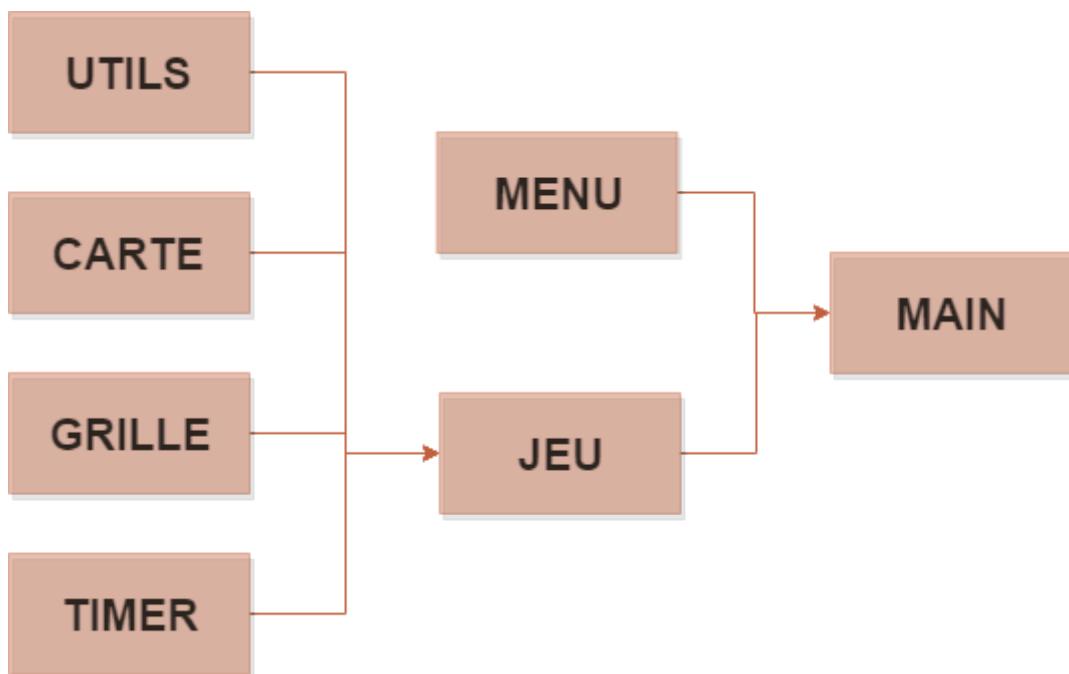
Ainsi, le sujet nous demandait de réaliser un jeu du memory : plusieurs cartes s’affichent, on les retourne 2 à 2 et on doit constituer des paires. Le sujet nous demandais aussi d’autres fonctionnalités : intégrer un timer, un mode triche, un menu. Des éléments qui complique la chose mais qui permettent au jeu d’être plus complet.

Ainsi dans ce document vous pourrez suivre le déroulement du projet et son fonctionnement, les difficultés rencontrées etc.

CONCEPTION

Structure

Pour simplifier la lecture du programme on l'a réparti en plusieurs fichiers contenant des fonctions suivant le schéma ci-dessous :



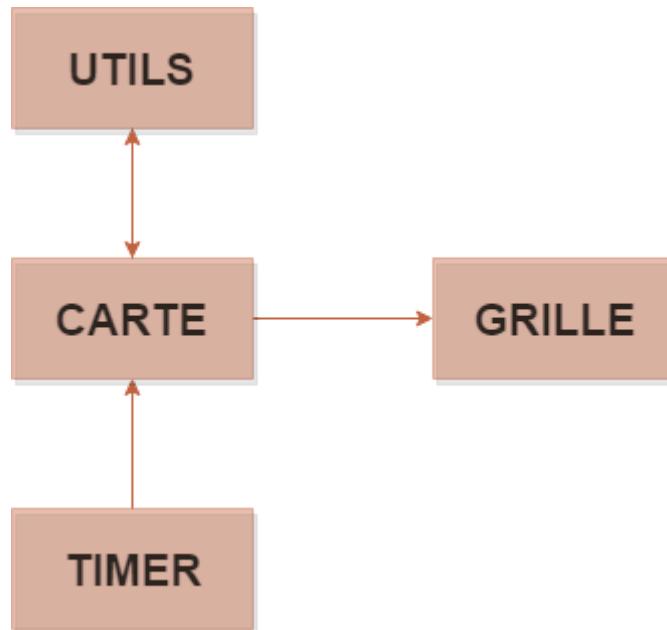
Le main : ce fichier a pour but de lancer le jeu en pré-définissant la fenêtre qui va s'ouvrir.

Le menu : ce fichier contient le code du menu principal, à l'intérieur on va retrouver le choix de niveaux (facile, moyen, difficile et aléatoire), les instructions et permet de lancer le jeu.

Le Jeu : ce fichier constitue le jeu en lui-même.

En ce qui concerne utils, carte, grille et timer, il s'agit de fichiers répertoriant plusieurs fonctions ayant chacun un thème de fonctionnalité.

Chacun de ces fichiers sont organisés de la manière suivante :



Timer : ce fichier contient les fonctions qui codent le timer du jeu (le temps écoulé depuis le début de la partie), il est utilisé par certaines fonctions de carte (pour l'affichage des cartes).

Utils: ce fichier contient des fonction diverses/générales, comme le survol de la souris au dessus des cartes, le mode debug...

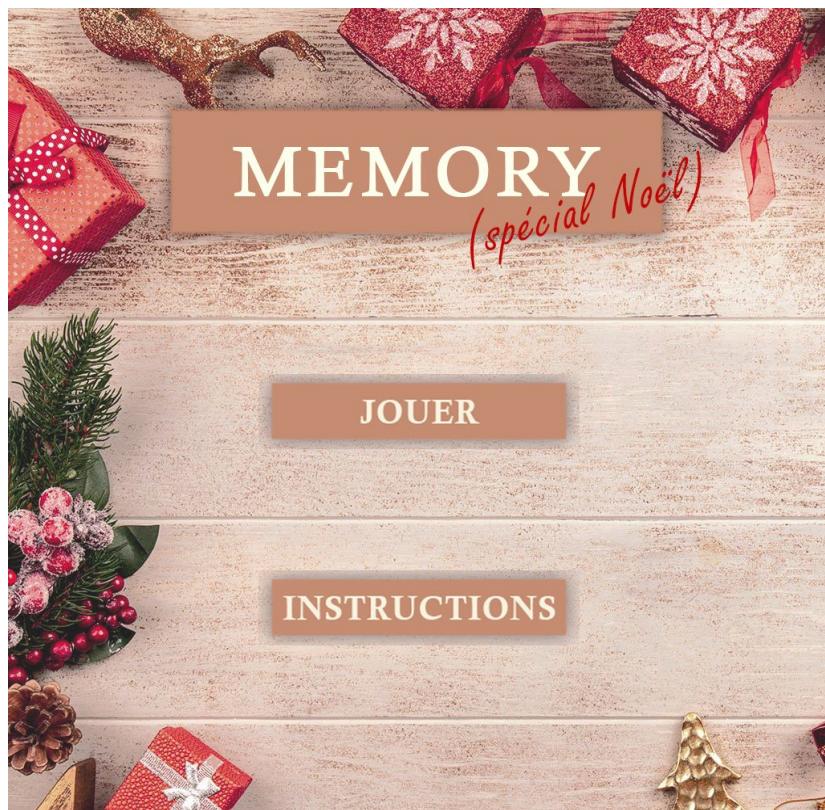
Carte : ce fichier contient les fonctions qui concernent le chargement des cartes qui sont ici des structures utilisant des sprites.

Grille : ce fichier contient les fonctions qui créent et remplissent les grilles d'ID (qui servent ensuite pour l'affichage des cartes/sprites) pour le jeu, affichent la grille, remplissent la grille de cartes à l'aide des id.

Fonctionnalités

- Menu :

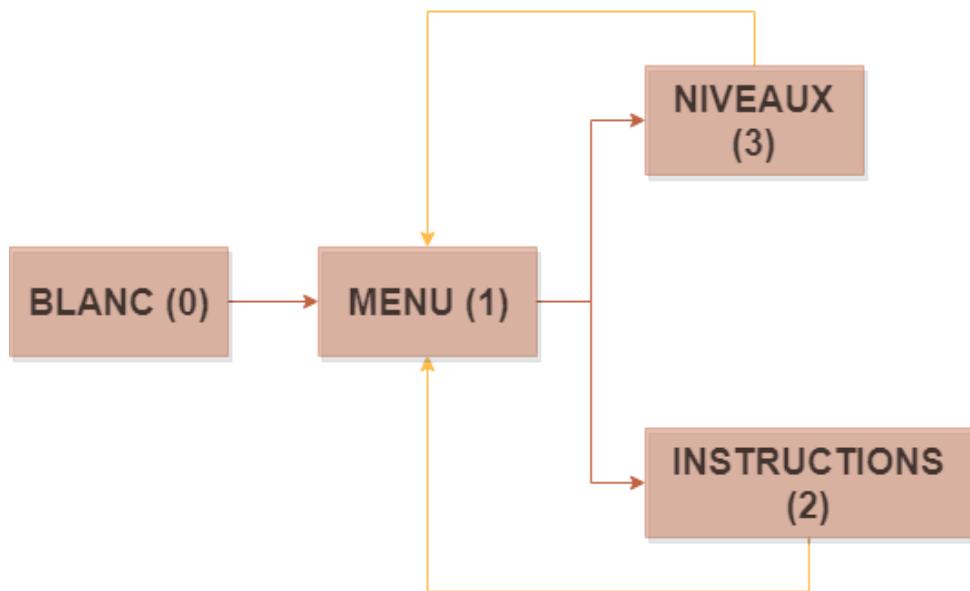
Au lancement du jeu, le joueur se retrouve au menu principal (codé par les fonctions contenues dans menu.c). Il peut alors faire deux choix : jouer ou lire les instructions :



Pour effectuer ce choix on a défini des zones sur le fond que l'on rend cliquables ; À chaque fois que le joueur clique dans une zone un nouvel écran apparaît et le joueur progresse dans le jeu.

Chaque écran est définie par un fond spécifique qui conduira le joueur à de nouveaux choix. Ces écrans s'affichent via des copie de zone, ce qui permet de ne pas utiliser une quantité de mémoire importante.

Ainsi le menu est décomposé en 4 écrans :



Blanc : écran vierge, permet de réinitialiser l'affichage.

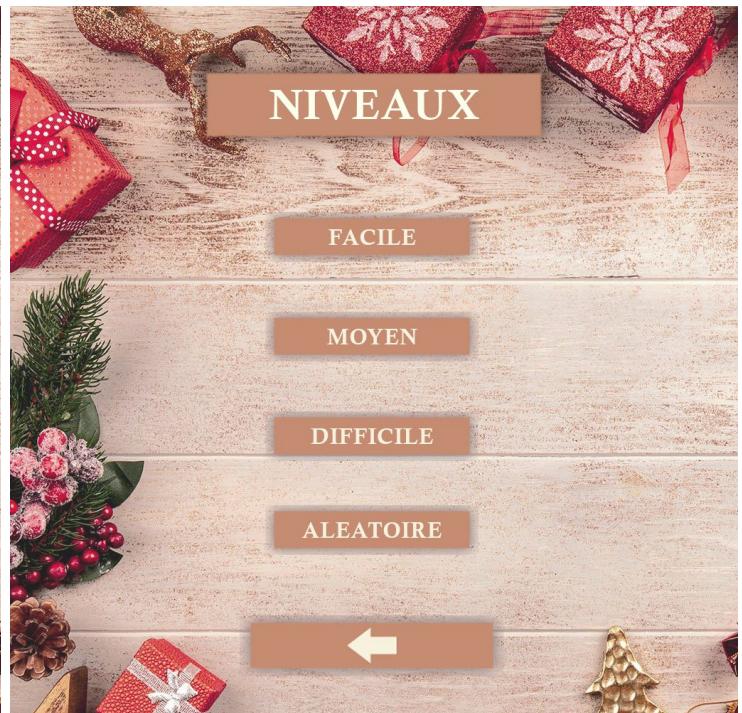
Menu : écran d'accueil, propose de jouer ou d'aller lire les instructions.

Instructions : explique les règles du jeu et permet de retourner en arrière vers l'écran menu.

Niveaux : propose au joueur de choisir entre 4 niveaux de difficulté différentes : facile, moyen, difficile ou aléatoire (ce mode charge un nombre de lignes aléatoirement). En fonction du choix, la grille de carte sera plus ou moins grande. Permet aussi de retourner en arrière vers l'écran menu.



écran instructions (2)



écran niveaux (3)

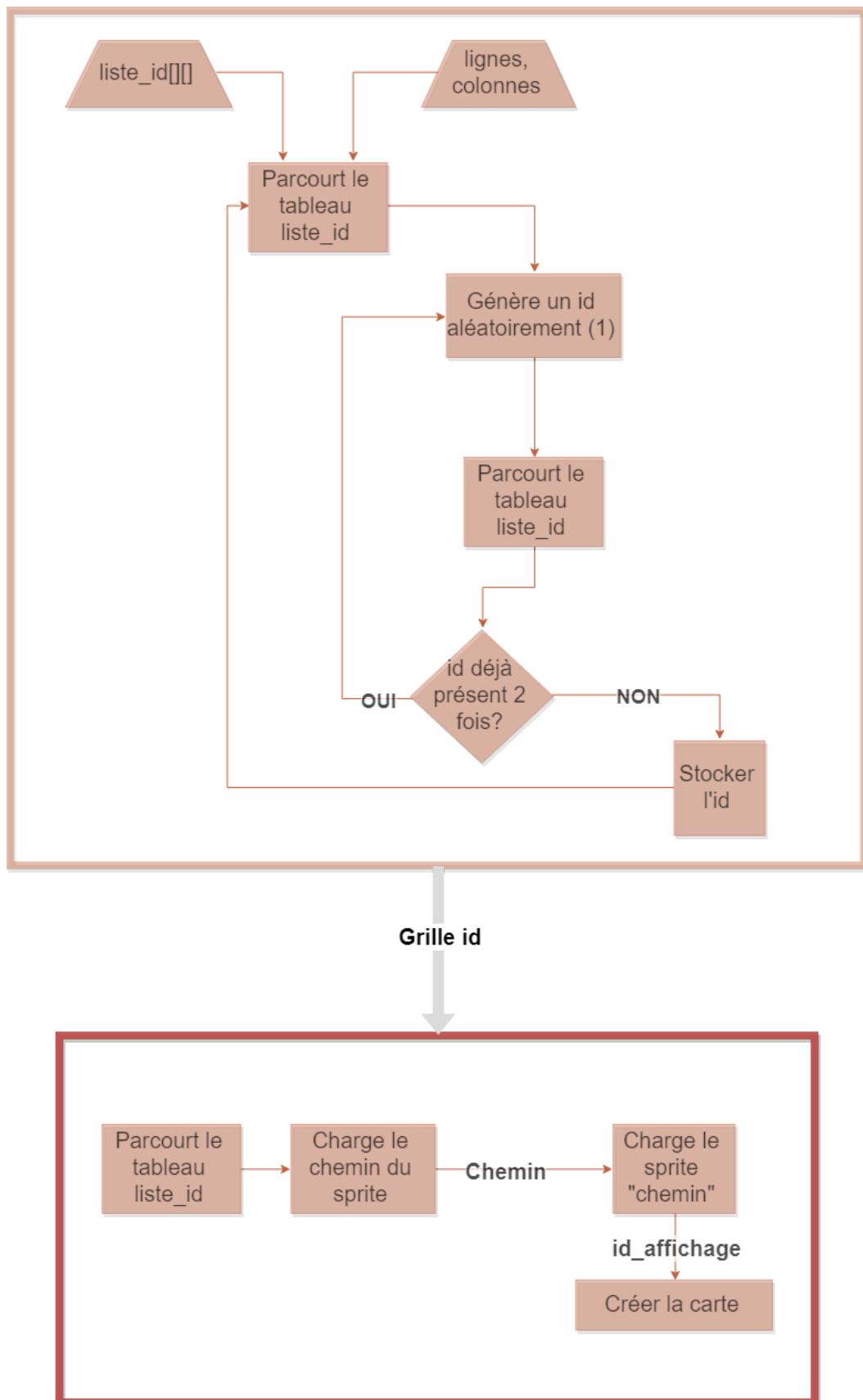
Afin de pouvoir rendre les "retour" possibles, on a ajouté une variable d'état (ici `etat_menu`) qui prend la valeur d'un entier en fonction du fond affiché. Ainsi, cette valeur permet d'afficher le fond correspondant et donc à l'utilisateur de naviguer entre les différents menu.

● Grille :

Pour que le jeu puisse fonctionner, il est essentiel d'utiliser des tableaux. En effet, on a besoin de répartir aléatoirement des cartes et de pouvoir comparer leur positionnement à partir du moment où deux sont retournées.

On sait que les cartes doivent avoir plusieurs paramètres : leurs positions, leur ID, si elles sont retournées ou non etc. Ainsi, afin de simplifier leur affichage, on a défini un type carte en utilisant une structure. Ce type sera par la suite utilisé dans la création des grilles pour y créer et stocker ces types cartes à la manière d'instances d'objets.

Pour la création des grilles, on a utilisé plusieurs fonctions (contenues dans le fichier grille.c), s'activant selon cet algorithme :



- (1) Génère un id aléatoirement entre 1 et la moitié du nombre de cases
 - (2) Stocker l'id dans la case actuelle du tableau liste_id[][]
 - (3) Charge le chemin du sprite dont le n° est la case liste_id[i][j]
 - (4) Créer la carte dont l'id est la case liste_id[i][j] et dont l'id d'affichage et id affichage.
- Positionnement selon multiple de l'index du tableau

RemplirGrilleID() : À l'aide de boucles, des ID de cartes vont être générés, leur nombre dépendra du nombre de lignes générées suite au choix de difficulté. Il leur sera ensuite attribué une position prédéfinie (les variables des boucles) dans un tableau à 2 dimensions. La fonction vérifie aussi que les ID se trouvent exactement 2 fois dans le tableau.

RemplirGrilleCarte() : fonction qui attribue les ID aux cartes et qui permet de créer les cartes ainsi que charger le nombre de sprite exacte. La position des cartes est définie automatiquement à l'aide de multiple des variables des boucles ajoutés à la position d'origine en abscisse et en ordonnée.

AfficherGrilleCarte() : permet d'afficher la face ou le dos de la carte en fonction du clic, si la carte n'est pas cliquée alors la fonction affiche le dos. La fonction possède un paramètre "mode_triche_activee" prenant la valeur de la variable pause de jeu.c (1 (vrai) ou 0 (faux)), si elle est égale à 1, toutes les cartes sont découvertes.

On a aussi ajouté la fonction LibererSpriteGrille() afin de libérer la mémoire occupée par les sprite lors du retour au menu principal ou en cas de victoire.

- Timer :

Afin de donner une idée du score au joueur, on a ajouté un chronomètre (appelé timer). Pour ce faire, on a dû créer une variable qui prend la valeur du temps donnée par la fonction **Microsecondes()** (de la bibliothèque graphique) lorsque le timer est initialisé, et la soustraire au temps donné actuellement par la fonction **Microsecondes()**. Après une conversion en seconde, on obtient ainsi le temps écoulé entre les deux compteur, donc des secondes .

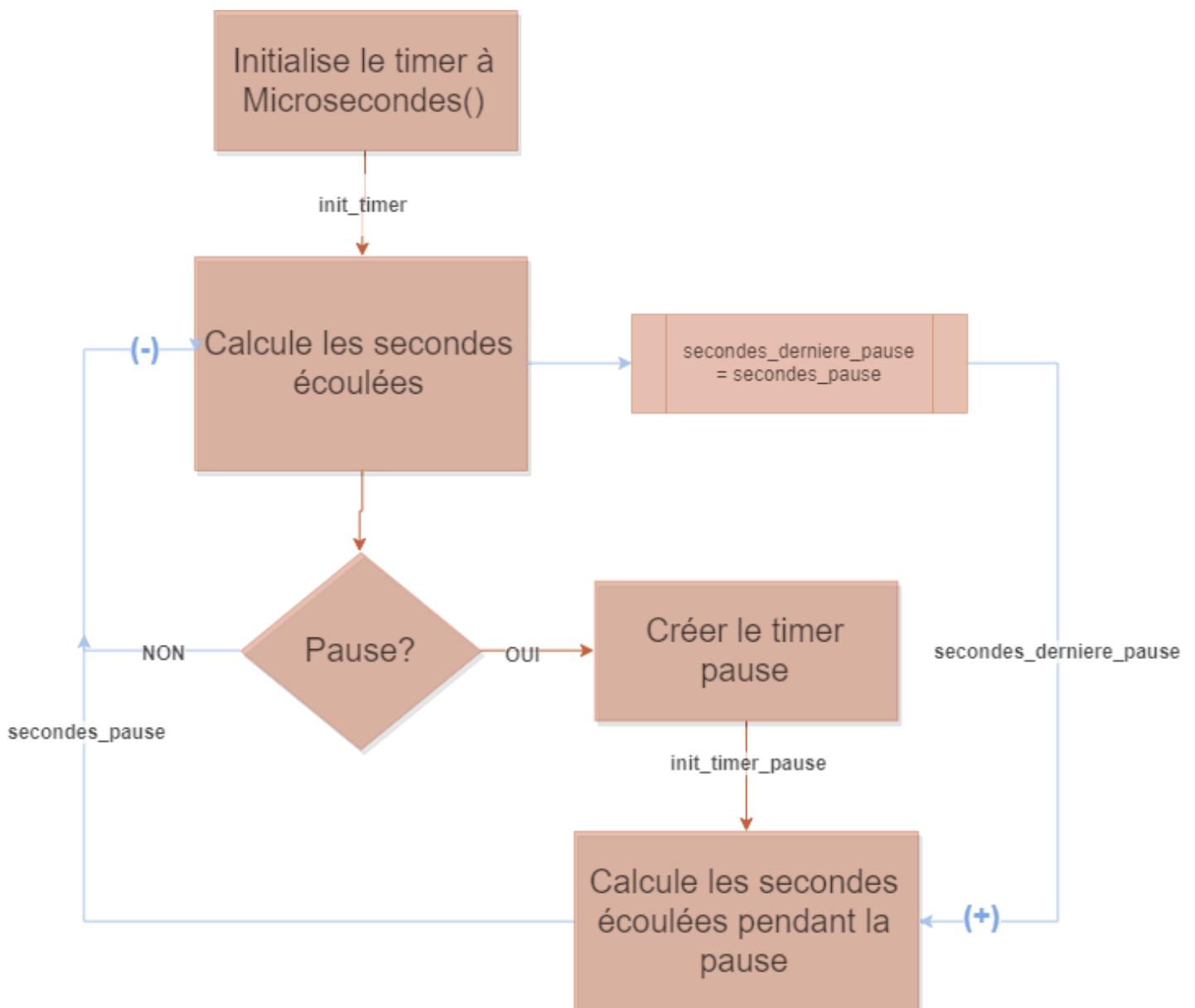


L'affichage du timer posait problème. En effet à chaque seconde écoulée le texte s'empile avec le précédent. Pour corriger ce soucis, on charge le fond dans la boucle *while* du jeu.

Pour optimiser l'utilisation de la mémoire on aurait pu faire un copier de zone juste dans la zone du timer.

- Mode triche :

Le mode triche met en pause le timer et affiche toutes les faces des cartes tant que la pause est activée. Pour faire cela, on a créé un second timer qui va enregistrer le nombre de secondes écoulées depuis le début du mode triche, et on va soustraire la valeur du second timer au premier.



Ensuite, pour afficher toutes les faces des cartes, lors de l'appui sur la touche "t", la variable *pause* (de la fonction *AfficherGrilleCarte()*) prend la valeur 1, donc elle affichera les faces de toutes les cartes. Si la touche est appuyée une seconde fois alors *pause* reprend sa valeur initiale (0, à la manière d'un booléen) et les cartes sont affichée de dos. Le mode triche active aussi un mode "debug", qui affiche la grille des id sur le terminal.

-----	-----	-----	-----	-----	-----	-----	-----	-----
3	1	1	6	4	4	3	7	
7	8	8	2	5	6	2	5	

Pour aller plus loin

Afin de rendre notre jeu plus plaisant, on a pensé à ajouter plusieurs fonctionnalités optionnelles.

- **Menu victoire :**

À tout jeu, ses victoires et ses défaites. Afin de bien indiquer au joueur qu'il a fini sa partie et qu'il a gagné, on a créé un écran de victoire qui annonce le temps écoulé lors de la partie et propose de rejouer.



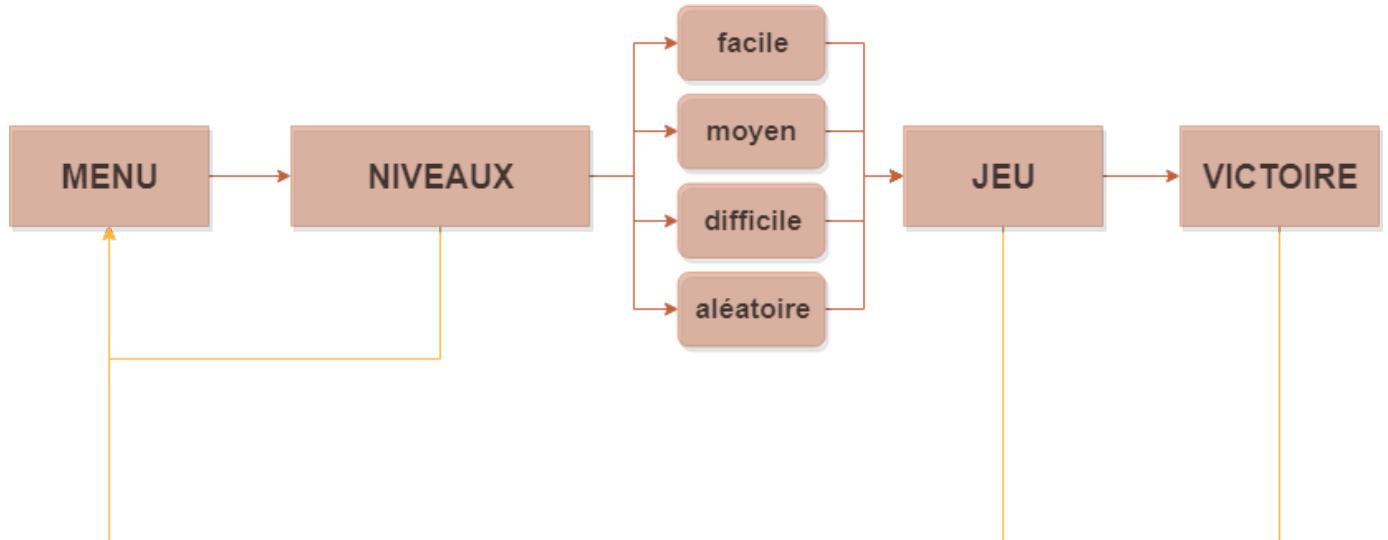
Ainsi on a créé une fonction `TestVictoire()` qui va vérifier que toutes les cartes qui ont été cliquées sont bien toutes retournées. Si tel est le cas alors on charge le fond de victoire. Une variable va alors prendre la valeur du timer et l'afficher dans la zone demandée.

Afin d'aller jusqu'au bout de la victoire, on a souhaité rajouter un système d'enregistrement des temps afin d'effectuer un classement. En utilisant les fonctions qui traitent des fichiers comme des flux (comme `fopen()`, `fread()`...).

Mais par manque de temps, on a pas pu mener cela à terme... Mais ça sera peut être pour une v2.

- Les retour :

A partir du moment où l'on doit naviguer dans une interface, il est important de pouvoir revenir au point de départ pour ne pas se perdre.



Ainsi on a ajouté des boutons de retour à chaque écran du menu (cf II) 1) Structure) et aussi lors du déroulement du jeu (pour ceux qui voudraient changer de niveaux en cours de route). Et pour finir : dans le menu victoire, afin de pouvoir relancer une nouvelle partie (ici le retour est désigné par le bouton "rejouer").

- Curseur :

Pour apporter une petite touche qui va dans le thème de notre jeu (et pour explorer la bibliothèque de fonctions graphiques), on a décidé de modifier le curseur de la souris de l'utilisateur.



On souhaitait aussi faire en sorte que la souris change lors du survole d'un bouton, voir de rendre le bouton dynamique lors du survole ou du clique de la souris.

Malheureusement, dû à la boucle while qui permet l'affichage constant du jeu, nous n'avons pas réussi à mettre en place ces fonctionnalités.

CONCLUSIONS

- Marie Pellier :

Réaliser ce projet fut intéressant pour moi car il s'agissait de produire quelque chose de concret. De plus, l'aspect graphique apporté par le jeu est quelque chose que j'ai plutôt apprécié. En effet cela permet de voir le résultat d'une programmation de manière dynamique.

Cependant, j'ai rencontré quelques soucis. Moi et mon camarade nous n'avons pas assez bien réparti le travail à effectuer, si bien qu'il a effectué beaucoup plus de chose que moi. De plus, la différence de niveau entre lui et moi fait qu'il me fallait plus de temps de réflexion et d'exécution que lui pour obtenir un résultat satisfaisant. J'ai pu tout de même l'assister, en apportant mon point de vue sur le comment procéder, comment améliorer le déjà effectué etc. Ainsi, je pense qu'à l'avenir, pour les prochains projets, je veillerai à mieux répartir les tâches afin que tout le monde y trouve son compte.

- Aurélien Peden :

Ce travail m'a permis pour la première fois de réaliser un projet informatique en groupe. Ce n'est pas la première fois que je réalise un jeu ou un programme, mais le faire en équipe a été enrichissant du point de vue de la gestion du projet et de la répartition des tâches (à l'aide de Trello), bien que cela soit perfectible comme l'a dit ma camarade. Le travail en commun m'a permis de pouvoir discuter de nos divers points de vue

quant à la conception de ce jeu, qu'ils soient par rapport à l'aspect technique du projet, de l'implémentation des fonctionnalités ou du jeu en lui-même.

Concernant les choses que j'aurais pu améliorés, j'aurais pu essayer de plus m'adapter au rythme de ma camarade, mes projets personnels m'ayant habitué à plus travailler à mon rythme qu'à celui des autres, étant d'ailleurs la raison pour laquelle j'ai décidé de réaliser ce projet en binôme.