IUT Sénart-Fontainebleau

RAPPORT PT1.1 APL 2018

Département
INFORMATIQUE

Table des matières

Introduction :

Nous devions réaliser pour ce projet un memory :

Un memory est un jeu de carte comportant des paires. Chacune de ces paires portent des illustrations identiques. Ces cartes sont distribuées aléatoirement face cachée à chaque partie afin que l'utilisateur puisse travailler sa mémorisation, ce qui est le but principal de ce jeu ludique. Le joueur peut ensuite retourner une première carte puis, en révéler une seconde, tout en laissant face recto la première carte. Différents scénarios se présentent alors face à nous:

- Si elles sont identiques, alors les deux restent découvertes.
- Si elles sont différentes, alors elles se retournent face cachée après 1 seconde de mémorisation pendant lequel on ne peut pas jouer.

Les fonctionnalités que l'on devait apporter à notre programme étaient les suivantes :

- La capacité de retourner les cartes par un clic de souris
- Afficher le temps (en secondes) écoulé depuis le début de la partie.
- On devait pouvoir jouer sur des grilles de tailles différentes.
- Le laps de temps de mémorisation d'une paire de cartes découvertes devait durer une seconde.
- Un mode tricheur devait être accessible en pressant la touche t qui découvre la grille et interrompait le temps jusqu'à ce que l'on rappuie sur cette même touche pour reprendre la partie.

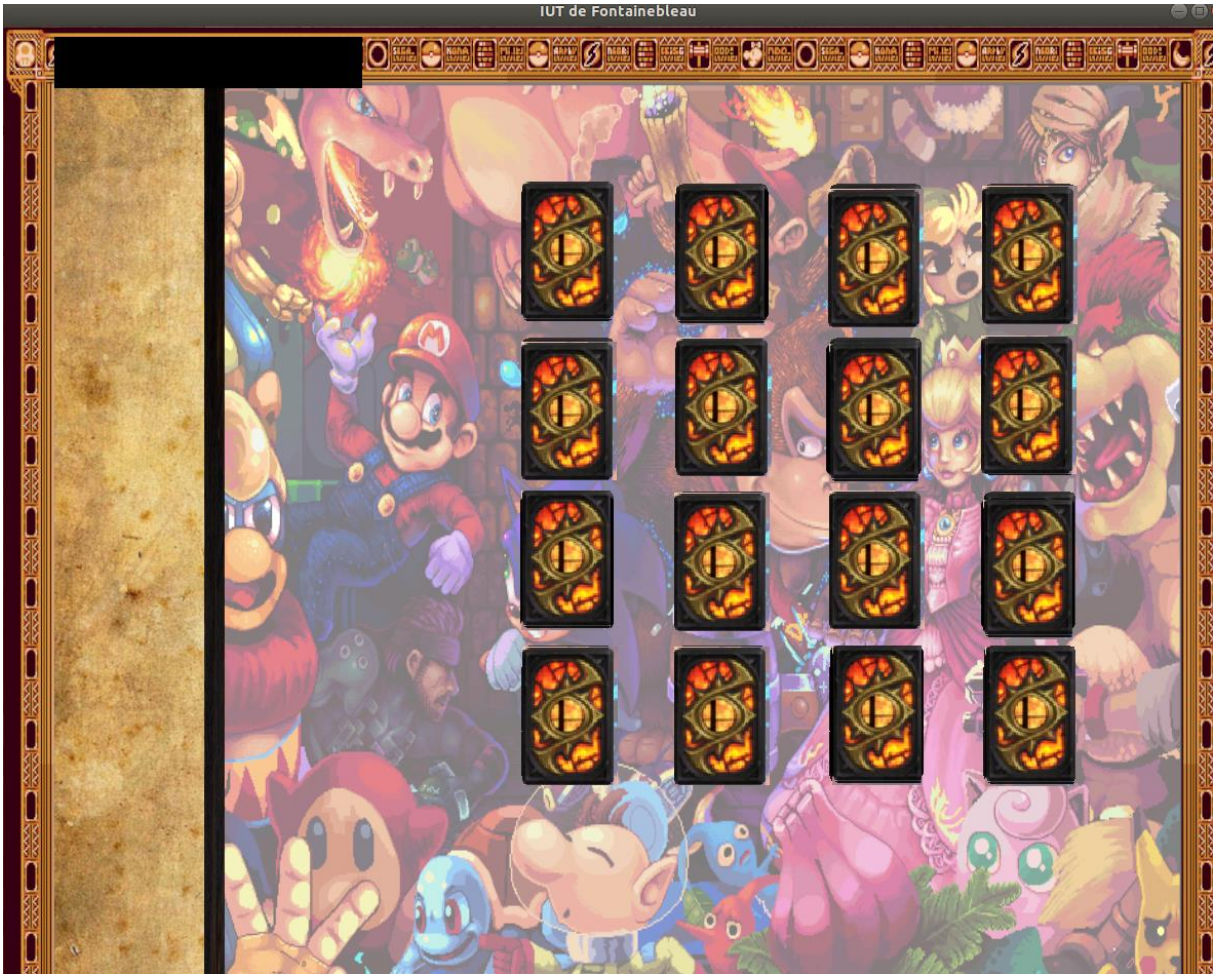
Notre travail devait être fournie dans un délai imparti, et sous la contrainte de certains formats (makefile, PDF, …)

Afin de faire honneur à l'un des jeux les plus cultes de notre génération, et suite à sa sortie récente, nous avons voulu rattacher notre loisir avec notre travail, c'est pourquoi nous avons dédié ce memory à Super Smash Brawl Ultimate. A travers notre memory, vous allez totalement plonger dans l'incroyable univers de ce jeu de combat mettant en scène des personnages iconiques de la saga.

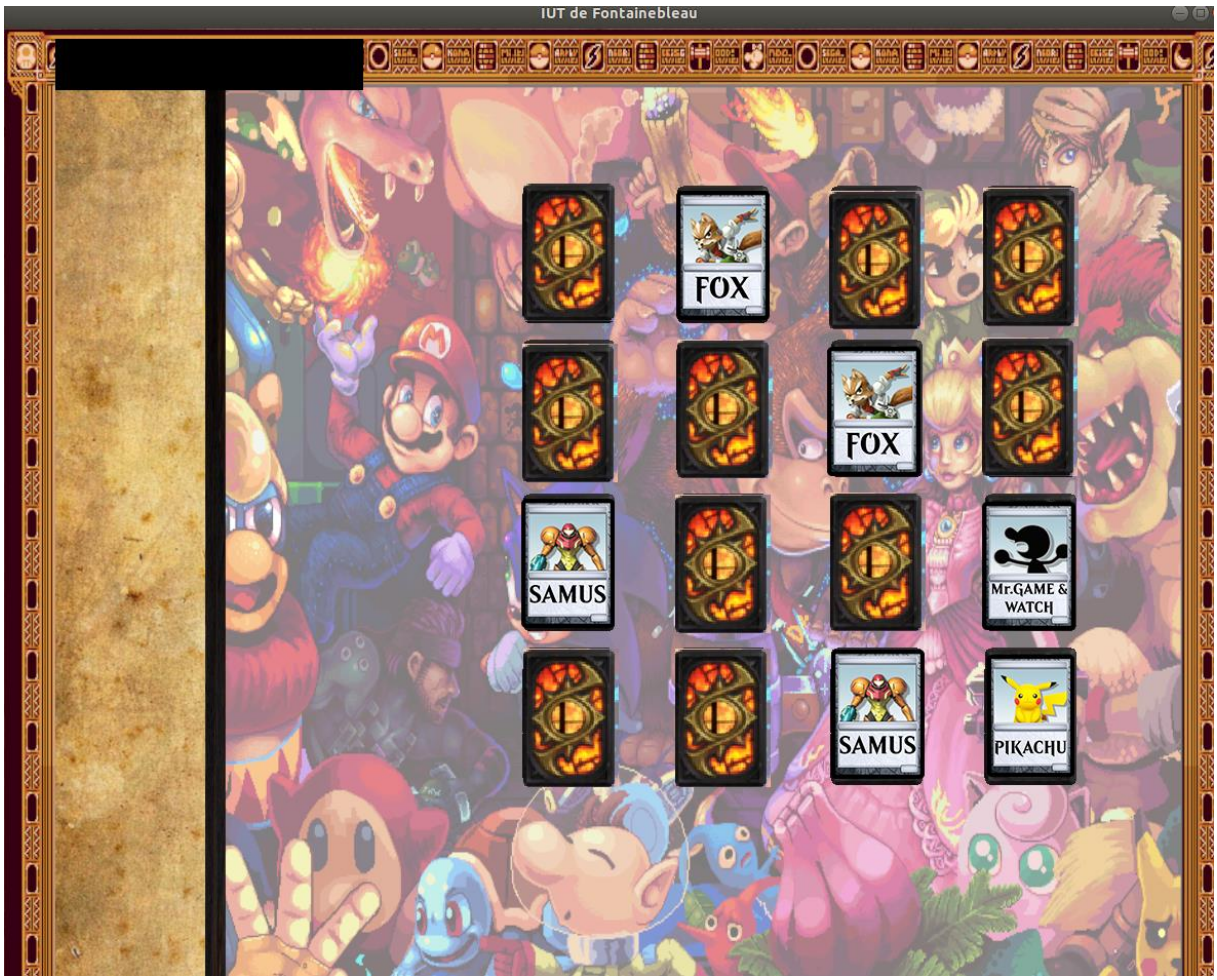
Réponse à la demande de fonctionnalités :

Clic :

Pour répondre à la demande de cette fonctionnalité, nous avons réussi à faire en sorte que les cartes apparaissent sur notre plateau de jeu face cachée et selon un nombre qui dépend de la difficulté prise par le joueur.



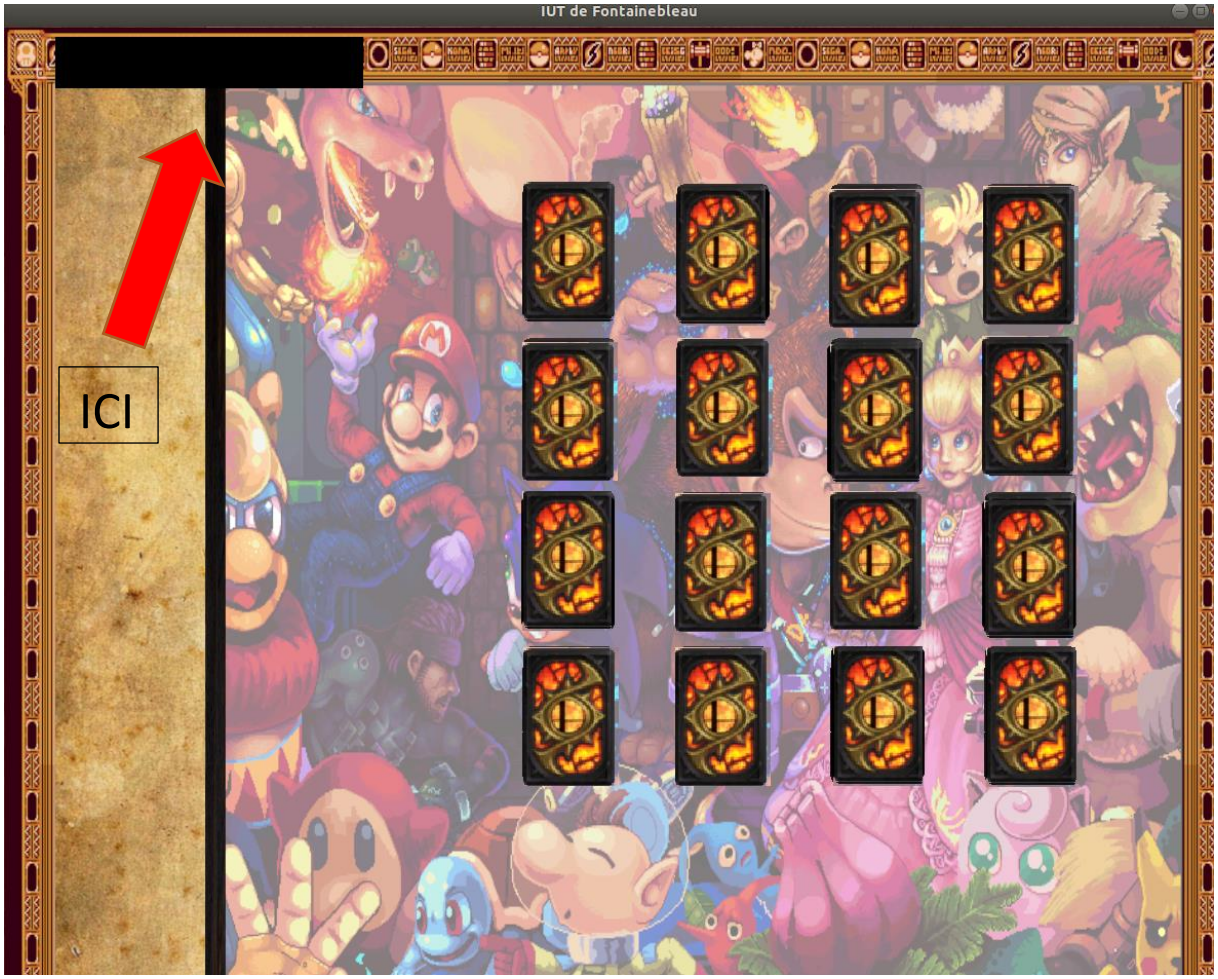
Puis à l'aide d'un clic de la souris, nous pouvons retourner ces cartes afin de voir les differents illustrations des cartes et de pouvoir tenter de gagner en trouvant toutes les paires.



Comme vous pouvez le voir avec cette photo, nous avons pu retourner de nombreuses cartes jusqu'à en trouver deux paires qui sont celles de Samus et de Fox.

Timer :

En ce qui concerne le timer, nous avons pu écrire sa fonction au sein de du programme jeu.c, mais celui-ci ne semble pas fonctionner correctement. Celui-ci devait apparaître dans la case noir en haut à gauche que vous pouvez voir dans la photo de ci-dessous.



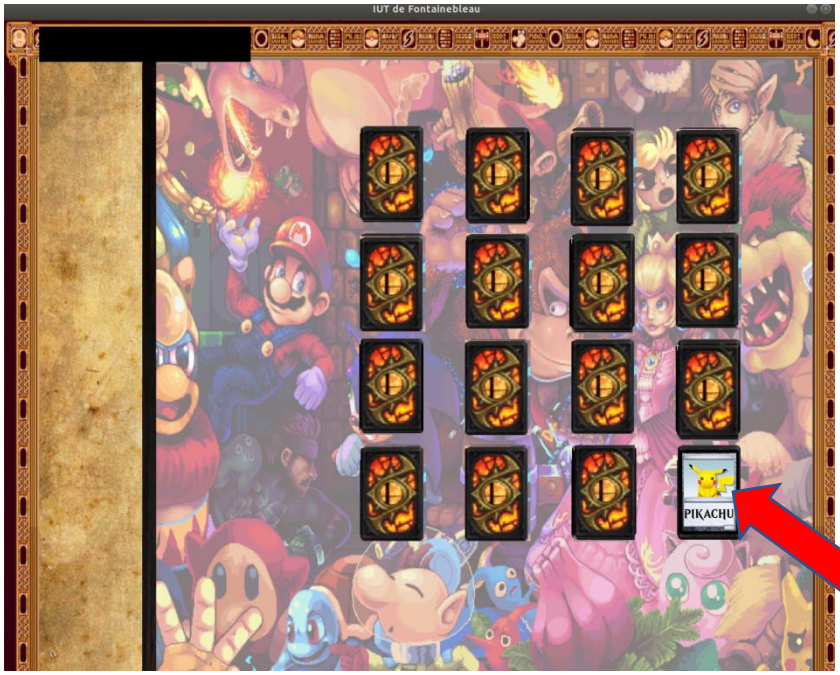
Pourtant, celui-ci se lançait par certaine occasion mais pas tout le temps comme on le voudrait.

Difficultés :

Pour répondre à la fonctionnalité concernant les grilles de tailles différentes nous avons choisi de proposer au joueur, plusieurs niveaux de difficultés plutôt qu'une grille réglable pour donner l'impression de jouer à un jeu classique. En effet, lorsque celui-ci accède au jeu, il fait face à ce menu où il doit choisir entre 3 niveaux de difficultés, allant du plus facile au plus dur. Ce qui change la difficulté d'un niveau par rapport à un autre, c'est le nombre de cartes. En effet, vous trouverez dans le niveau facile, un jeu de 8 paires, dans le niveau moyen, un jeu de 12 paires et pour ceux qui en ont le courage, la volonté mais aussi le temps, pourront se rassassier d'un jeu de 18 paires dans le niveau difficile.

Mémorisation :

Afin de répondre à la fonctionnalité du temps de mémorisation d'une seconde, nous avons dû utiliser la fonctionnalité sleep(1) qui permet au programme de "s'endormir" durant 1 seconde. Vous pourrez trouver cette fonctionnalité dans la bibliothèque de "unistd.h" que nous avons pu déjà rencontrer lors de cours en ASR. Cette fonctionnalité nous est très utile dans notre jeu, en effet lorsque le joueur aura préalablement cliqué sur 2 différentes pour les retourner, si elles ne sont pas identiques, celles-ci resteront une seconde afin de permettre au joueur de pourvoir mémoriser l'illustration présente sur ces 2 cartes.



1er clic:

Retournement de la carte

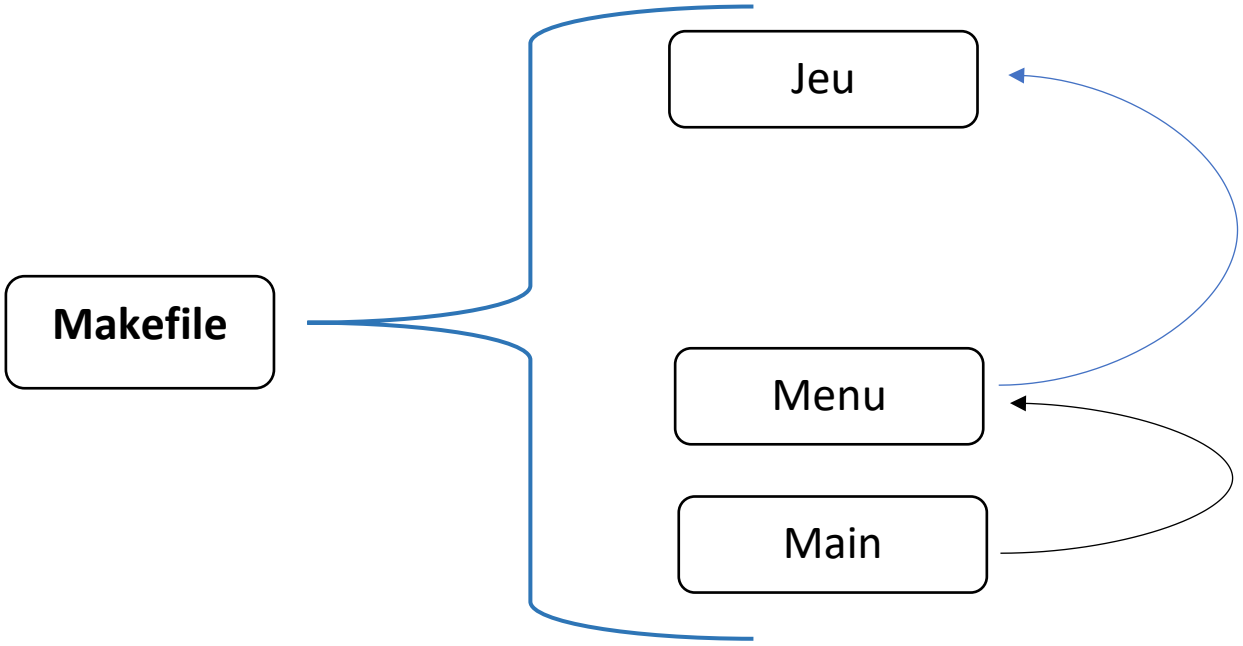
Mode tricheur :

En ce qui concerne le mode tricheur, il s'agit de la même situation que celui du timer, en effet, il marche dans certaines occasions et dans d'autre pas. Vous pouvez retrouver son programme dans jeu.c. Cela vient, je pense, d'une erreur d'appel de fonction qui fait que la fonction tricheur doit être utilisé après chaque vérification entre 2 cartes, sinan nous ne pouvons pas continuer à retourner les cartes.

Structure du programme :

Makefile :

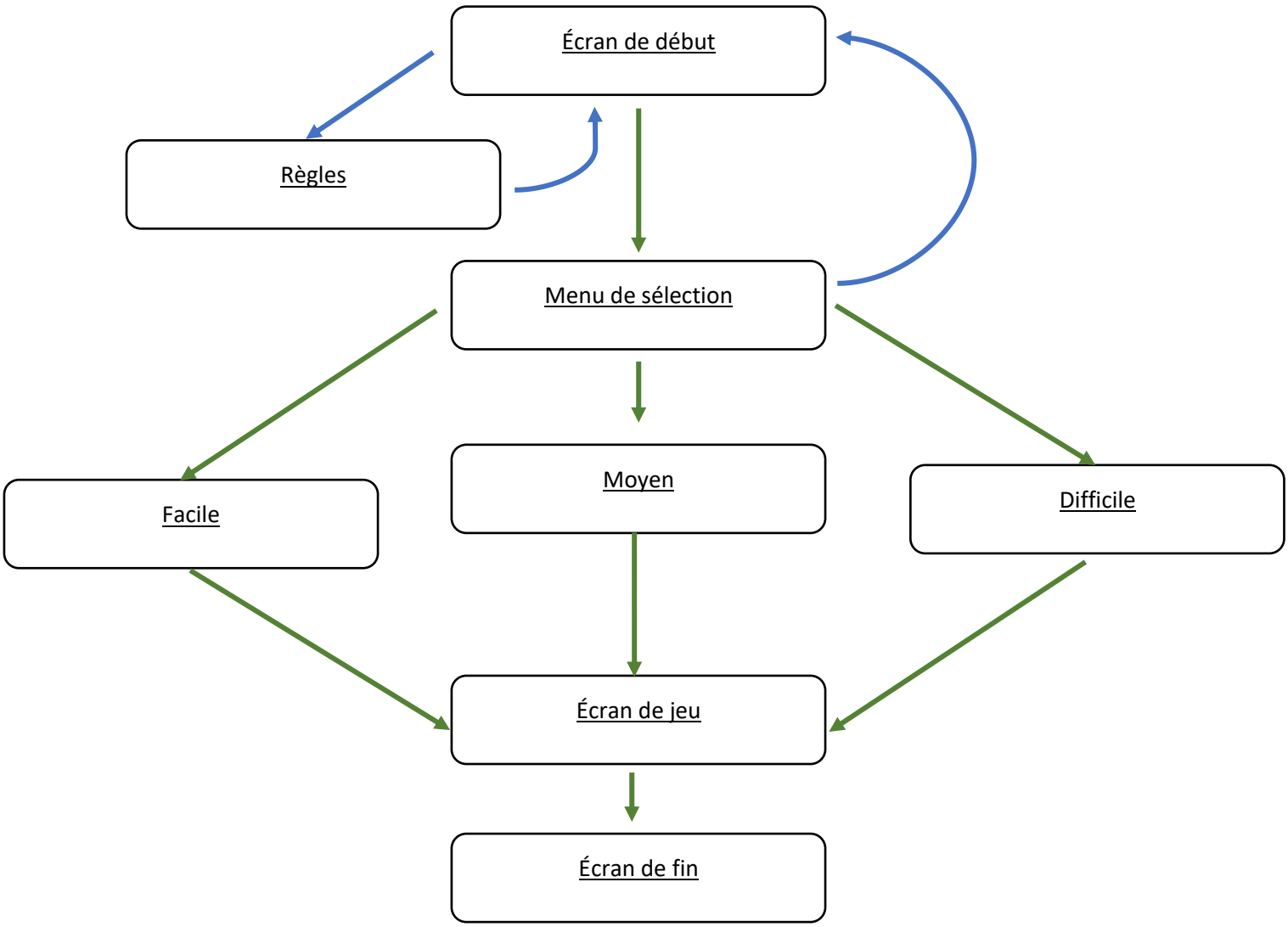
Le programme est construit à l'aide d'un makefile faisant appel à nos différentes fonctions. Les flèches ci-dessous représentent les dépendances.



Comme vous pouvez le voir à l'aide de ce schéma, nous avons dans notre jeu, 2 fonctions dans notre main. Celle-ci fait appel à la fonction menu qui fait elle-même appel à la fonction jeu.

Écrans :

Le jeu est structuré autour d'un enchainement de différents écrans accompagnant l'utilisateur pour faire ces choix et naviguer au travers du jeu.



Ce schéma représente les différents chemins que peut emprunter le joueur. De plus, lorsque celui-ci arrive à atteindre la fin d'un niveau, vous verrez apparaître un écran vous récompensant pour votre bravoure qui se poursuit par la fermeture automatique du jeu.

Écrans virtuels :

Nous avons aussi utilisé plusieurs écrans virtuels pour pouvoir manipuler les différentes données et fonctions avec plus d'efficacité. Ces écrans n'apparaissent pas dans la fenêtre graphique mais en arrière-plan lorsqu'ils sont appelés. La fenêtre graphique étant référé comme étant l'écran virtuel zéro.

Écran 1	Écran 2	Écran 3	Écran 4	Écran 5	Écran 6
Héberge le premier fond d'écran	Héberge le second fond d'écran	Héberge l'écran contenant les règles	Héberge les cartes face verso	Héberge les cartes face recto	Héberge un seul dos de carte à utiliser pour les fausses combinaisons

Nous utilisons aussi l'écran virtuel 9 afin de pourvoir sauvegarder la progression du joueur lorsque celui-ci utilise le mode tricheur.

Formes des données :

La fonction qui gère l'affichage du memory est *jeu_carte*, elle a besoin de l'entier *niveau* en argument, qui lui permet de moduler les valeurs de différentes variables. Dans cette fonction, on a aussi une variable *paire*, *ligne* et *colonne* :

Paire est un entier qui prend la valeur du nombre de paires relative au niveau.

Ligne est un entier qui prend la valeur du nombre de lignes relative au niveau.

Colonne est un entier qui prend la valeur du nombre de colonnes relative au niveau.

Max est une variable égale à *paire* * 2 nous permettant de randomiser correctement les cartes. Nous avons aussi deux variables contenant un entier pour les marges afin de centrer nos plateau de carte au centre : *marge_x* et *marge_y*.

Pour la génération aléatoire de notre tableau, nous avons utilisé une matrice afin de pouvoir identifier chaque carte à un entier spécifique.

tab_carte est un tableau de caractère contenant toute l'arborescence des différentes images utilisés avec la fonctionnalité snprintf qui me permet à ce moment-là de réutilisé la matrice que j'ai créé auparavant pour afficher mes différentes cartes dans l'écran virtuel 5.

Tandis que les tableaux d'entiers *random_facile*, *_moyen* et *_difficile* contiennent des entiers par paires pour qu'il soit ensuite randomiser.

Lorsque le joueur choisit une carte il est important de connaître les coordonnées i et j, qui seront stockés dans les variables x1, y1, x2, et y2, ils permettent de connaître de l'emplacement de cette carte afin de pouvoir de l'identifier avec son jumeau lors de la vérification.

Le programme n'accepte pas que le joueur choississe la même carte, en effet il lui est interdit de faire cela.

La variable score est une variable qui correspond à la limite de la boucle while. Lorsque le joueur découvre une paire, le score augmente de un jusqu'à atteindre le nombre de paire maximale. Lorsque ce nombre est atteint, le jeu est terminé.

Algorithme grille :

Afin de remplir notre grille en début de partie, nous avons tout d'abord créé une matrice selon le niveau de difficulté choisi par le joueur. En effet, si celui-ci choisit le niveau facile, alors une matrice 4x4 sera alors créée avec des données choisies aléatoirement dans le tableau random_facile={"1,1,2,2,3,3 etc" de sorte que la matrice ne contienne que des paires. Et donc pour remplir cette matrice, nous avons utilisé un programme qui pioche une case aléatoire de ce tableau et qui ensuite la supprime en décalant toutes les valeurs d'un cran. Ce qui permet par cette méthode de ne plus avoir de chance de retomber sur cette valeur. Une fois la matrice créée qui correspond dans mon programme à un plateau à 2 dimensions, i et j, il ne reste plus qu'à afficher les différentes cartes selon l'id qui leur est donné. Chaque carte possède un nom bien spécifique dans mon répertoire image tel que ./image/1.png. Grâce à la fonctionnalité de snprintf, nous avons pu créér un tableau contenant toute les arborescences. Il ne reste plus qu'à les afficher à l'aide de 2 boucle for et le tour est joué.

Conclusions personnelles :

J'ai trouvé ce projet très amusant et intéressant à réalisé car nous avons pu expérimenté le travail en équipe tout en travaillant sur quelque chose de ludique puisque le but ultime de projet était la réalisation d'un mini jeu. De plus, j'ai pu me rendre compte très vite, qu'il n'est pas si simple et facile à réalisé des choses dans le monde de l'informatique tel que le simple fait de provoquer le retournement d'une carte. En effet, nous avons souvent l'habitude d'être à la place du client qui utilise les applications, les logiciels, les jeux mais en fin de compte, on ne s'est jamais rendu compte réellement de l'ampleur du travail derrière tout sa. Je pense que ce n'est qu'en étant soi-même face au problème qu'on réalise la charge de boulot que peut demander le plus petit des programmes et je trouve ça fou. Pour finir, je pense que ce projet a été une belle occasion de se rendre compte de la charge de travail que nous attend pour la suite mais aussi de nous apprendre à savoir travailler de manière autonome.

Man Antoine

Bien que le projet était plutôt court, il m'a permis de découvrir le travail en équipe et les difficultés que l'on peut rencontrer. J'ai aussi expérimenter la pression qu'un tel devoir peut fournir et comment l'on peut gérer ceci. Ce fut aussi intéressant de découvrir les capacités graphique avec le langage C ! Le projet a été une bonne occasion de se rendre compte de la « vraie » manière de travailler et ceci m'a fait réfléchir sur mes méthodes de travail à un moment qui me paraît crucial

Noah Lacaste