BAAZIZ Majdi
QUERNEC Thomas

Rapport de jeu : Le Taquin
PT 1.1

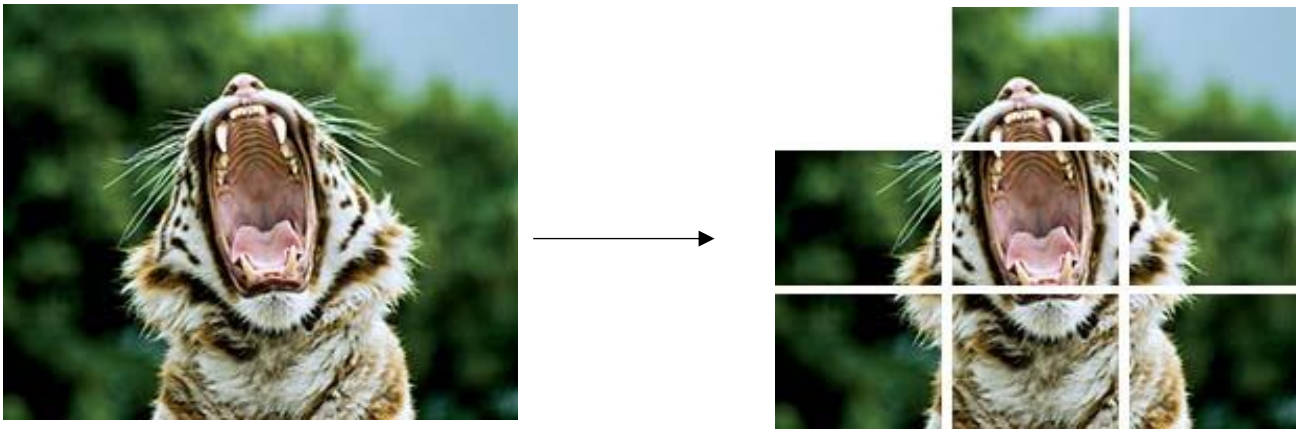
IUT Senart-Fontainebleau

I – Introduction

Le taquin, oui mais c'est quoi le taquin ? Le Taquin est un jeu solitaire créé vers les années 1870 aux Etats-Unis. Il a pour objectif de reformer une image préalablement divisée en plusieurs morceaux.

Dans notre cas, il nous a été demandé de concevoir ce jeu en langage C (C89) à l'aide d'une bibliothèque graphique. De plus certaines contraintes nous ont été formellement imposées :

- Le joueur doit pouvoir choisir une image parmi une sélection qui sera composée d'images toutes de résolutions différentes
- Le joueur doit pouvoir ensuite choisir indépendamment le nombre de lignes et de colonnes (entre 3 et 8)
- Les images vont être mélangées d'une façon à ce que le joueur a la possibilité de gagner c'est-à-dire de revenir au point de départ
- Le joueur doit pouvoir réaliser toutes les interactions que ce soit le menu ou le jeu à la souris et au clavier



(Source : iut-fbleau.fr)

II – Fonctionnalités

1) Police d'écriture et style

Avant de commencer à coder, on a murement réfléchi avec mon collègue sur la question de présentation du jeu. Puis nous avons décidé d'opter pour un « background » assez classique de telle sorte à ce que le système du jeu reste assez fluide. Mais nous avons tout de même voulu ajouter une touche de style. Pour commencer nous avons opté pour une police d'écriture « Graffiti ».

Cela permettait facilement de donner un effet de surbrillance au passage du curseur ou au clavier comme cela :

Remarque :

Il faut savoir que pour la sélection au clavier, la sélection est par défaut sur le bouton quitter.

Bouton sélectionné

Bouton non sélectionné

Le cas de la souris

Pour appliquer cet effet de surbrillance il suffisait juste, au passage de la souris à une certaine zone de coordonnées (du début de l'image à la fin) d'afficher une autre image par-dessus, il s'agit alors de la même image mais « remplie » puis lorsque la souris s'éloigne de cette zone, on réaffiche l'image de base « non-remplie ».

Quant au cas des images nous avons opté pour un remplissage extérieur c'est-à-dire qu'au passage de la souris sur l'image on verra apparaître un « tour-rectangle » (c'est un mot que je viens d'inventer pour dire que c'est une association de quatre rectangle qui permettent d'entourer un rectangle) de couleur différente du fond (ici nous avons choisi le cyan), le système est alors le même que pour les boutons, au passage du curseur dans la zone de coordonnées de l'image le « tour-rectangle » apparaîtra puis disparaître par une surcouche d'un « tour-rectangle » de la même couleur que le fond sur l'ancien « tour-rectangle ».

De plus nous avons ajouté l'affichage des résolutions de chaque image (sauf pour celle de l'image mystère) au passage du curseur.

(image)

Le cas du clavier

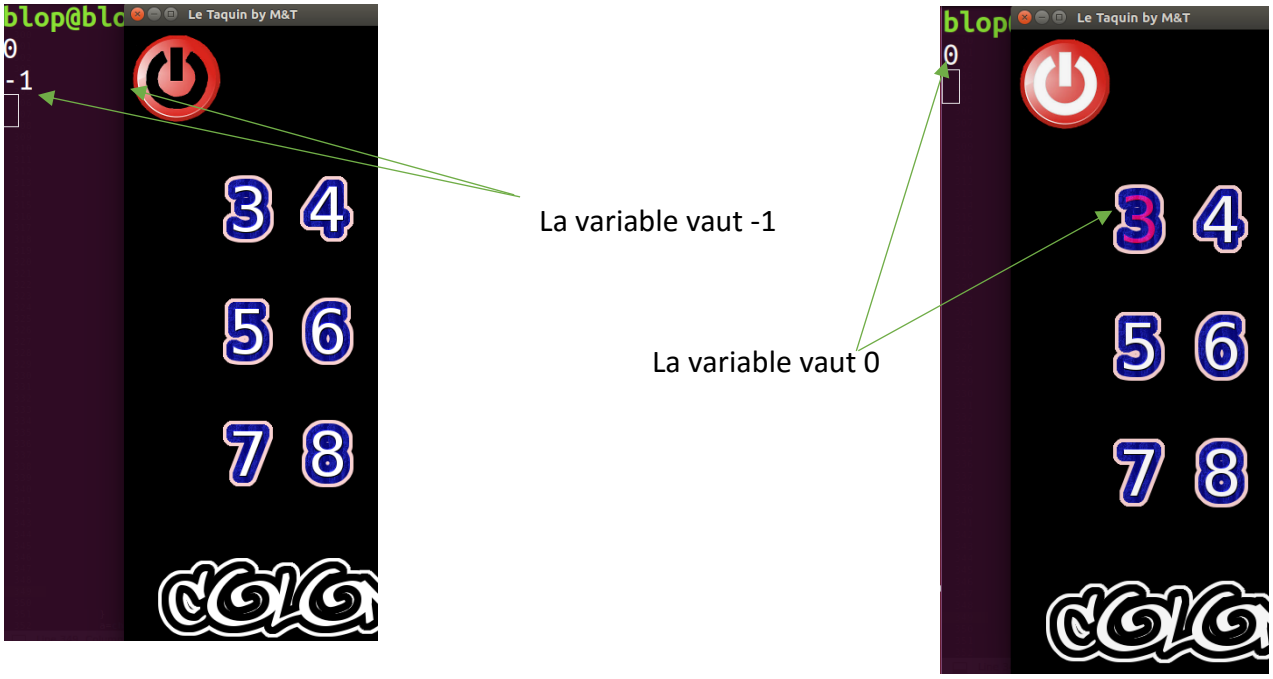
Pour gérer la surbrillance avec le clavier, nous avons décidé d'utiliser un système de variable « volante » puis nous avons mis dans les conditions d'affichage, en plus des zones de coordonnées pour le curseur, une certaine valeur qui sera suffisante à l'affichage de la surbrillance. Dès que le joueur appuiera sur une flèche directionnelle, la variable va soit s'incrémenté ou se décrémenté ou s'augmenter de 2 ou se soustraire de 2, cela dépendra des cas, en effet pour les cas où il y a plusieurs colonnes de choix tel que pour le choix des images ou le choix du nombre de colonnes et de lignes la variable pourra alors augmenter de 2 ou se soustraire de 2 mais pour les fenêtres contenant que 1 ou 2 boutons, la variable pourra que s'incrémenter ou se décrémenté. Et alors si la valeur de la variable rempli une des conditions, on verra alors voir la surbrillance de l'image liée à la condition. Et pour les cas où la variable n'est pas égale aux conditions, on affiche les images « dé-surbrillé » liées à toutes les conditions qui ne sont pas remplies.

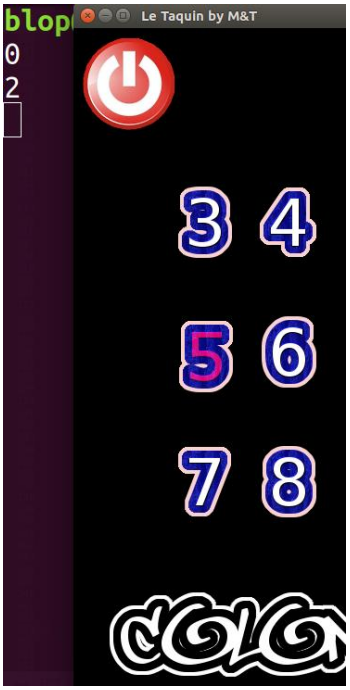
Exemple :

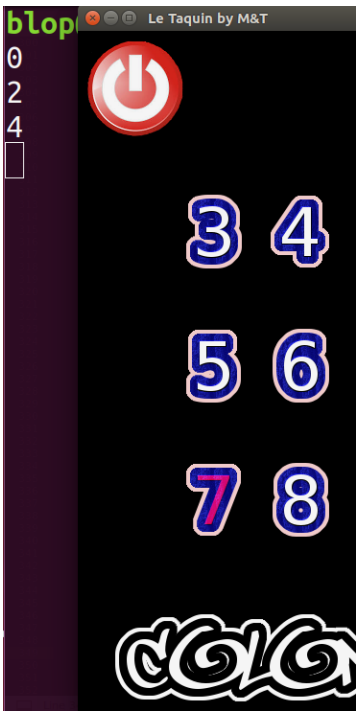
Pour le cas du choix des colonnes et des lignes, nous avons la variable volante x que nous avons initialisé à -1. Comme nous l'avons dit dans la remarque précédemment le « curseur » du clavier est toujours placé par défaut sur le bouton quitter, donc -1 correspond à la valeur suffisante de la condition d'affichage du bouton quitter en surbrillance. Ensuite, si le joueur appuie sur la flèche droite, la variable va s'incrémenter (la flèche de gauche décrémente, celle de haut soustrait de 2, celle de bas augmente de 2) puis elle va remplir une des conditions correspondant à (x==0) et va afficher l'image lié à la condition, ici c'est la valeur « 3 » qui est mise en valeur, et du coup, cela met le bouton quitter dans l'état « dé-surbrillé » et ainsi de suite.

Remarque : La variable volante ne peut avoir que un certain nombre de valeurs qui correspond au nombre de boutons présent dans la fenêtre.

Voici des captures d'écran du programme en marche montrant ces changements de valeurs de la variable volante.







2) L'image Mystère

Dans la sélection des images, dans le coin bas-gauche, on peut voir une image marqué d'un point d'interrogation et qui à son survol n'affiche aucune résolution. Cela est due au fait que c'est une image aléatoire. Lorsque le joueur cliquera dessus et choisira ses colonnes puis lignes, il aura la surprise de voir une nouvelle image à chaque fois (s'il a de la chance, il peut tomber sur la même) qu'il relancera le programme. Cela est dû à une petite fonctionnalité que nous avons ajoutée. Dès que la personne cliquera sur l'image mystère, un nombre aléatoire va alors se généré (un nombre entre 0 et 3), et à chaque nombre y est associé une image.

III – Structure interne du programme

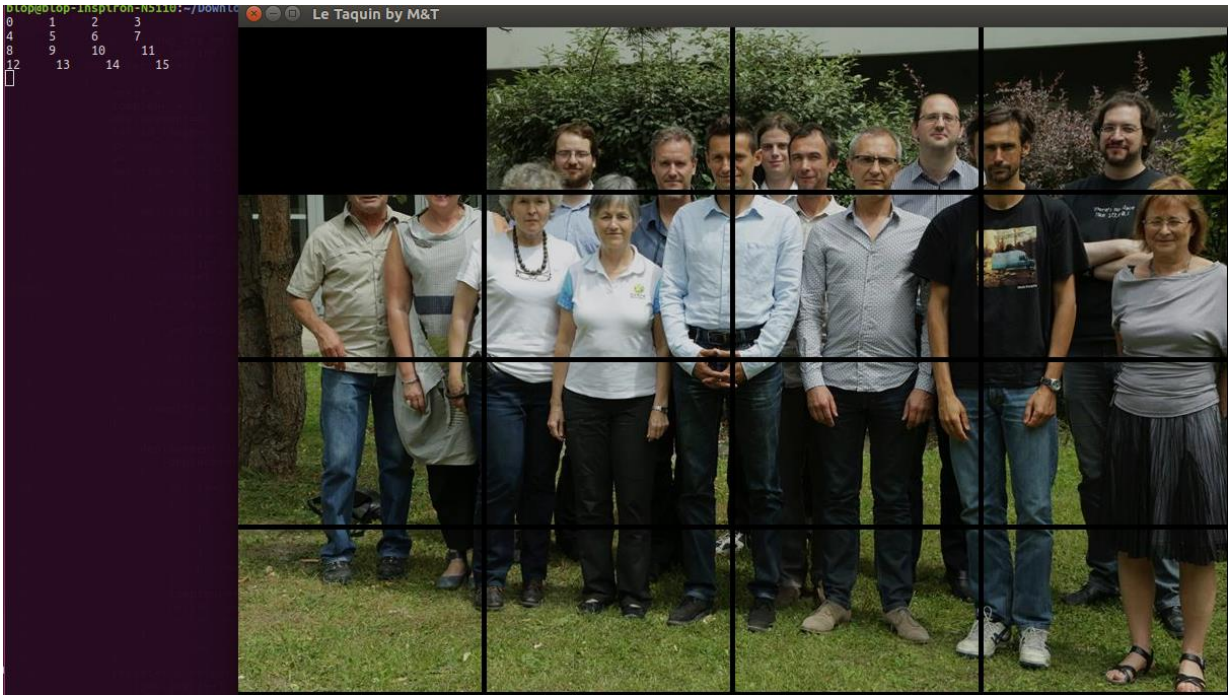
Pour structurer notre programme, nous avons pensé à un système progressif c'est-à-dire que une par une, les fonctions vont fournir des données (des informations plutôt), puis avec une autre fonction, nous combinons toutes ces données pour pouvoir lancer une autre fonction et ainsi de suite. On peut citer la combinaison des fonctions « *remplir_struct_img »* et *« concordance »*, en effet, la première permet la récolte de données qu'elle stocke dans une structure puis la deuxième traite ces informations récoltées et « ré-rempli » une structure avec la fonction « *remplir_struct »* ensuite la structure maîtresse renvoyée va permettre de mettre en commun la partie graphique avec la partie « tableau ».

Nous avons décidé de séparer nos fonctions par leurs domaines d'action. En effet, toutes les fonctions liées à la gestion des images et à la préparation à leur découpe sont situés dans un seul fichier intitulé « image.c ». Ensuite nous avons toute les fonctions liées au menu, c'est-à-dire le choix de l'image, des colonnes, des lignes, dans un seul fichier intitulé « menu.c ». Nous avons les fonctions liées au jeu lui-même c'est-à-dire le déplacement des tuiles, l'écran de fin, dans un seul fichier intitulé « jeu.c » et puis nous avons un fichier contenant toutes les fonctions annexes c'est-à-dire le remplissage du tableau, la vérification, la surbrillance des images (le « tour-rectangle » cyan), l'affichage des tentatives, intitulé « autre.c ».

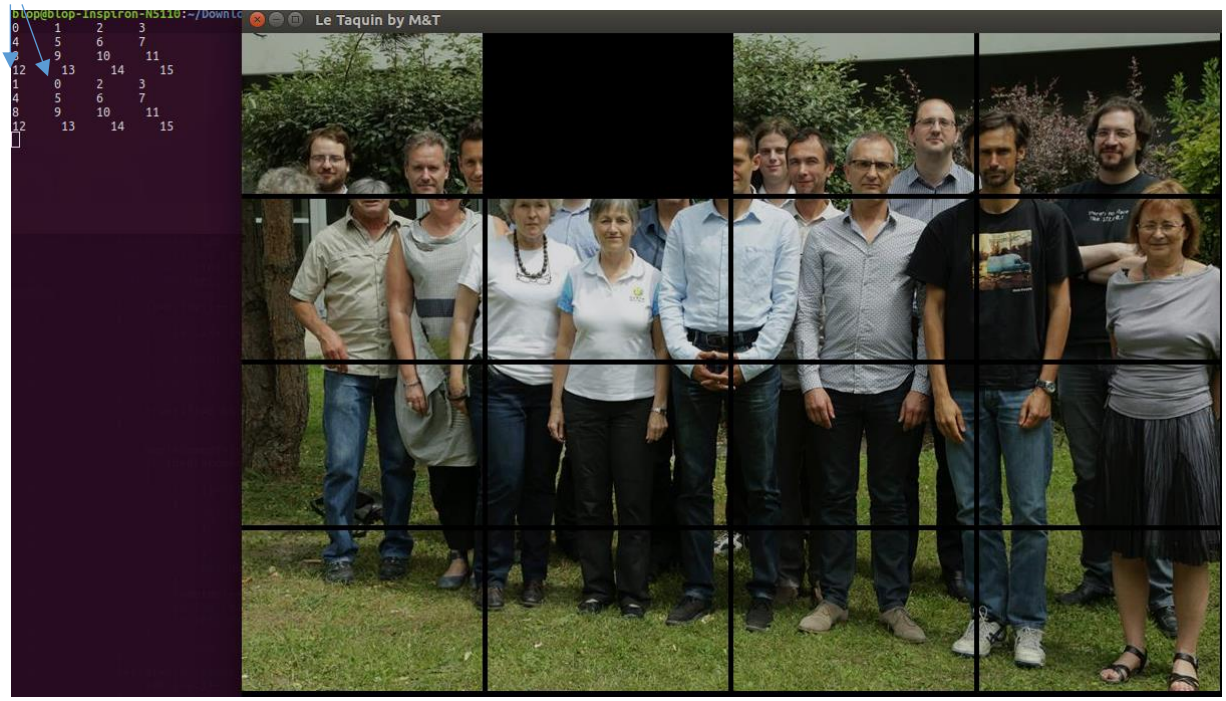
IV – Données d'une partie

Le jeu que nous avons codé est basé en partie sur système de tableau, c'est-à-dire que derrière chaque tuile, on trouvera une valeur d'un tableau.

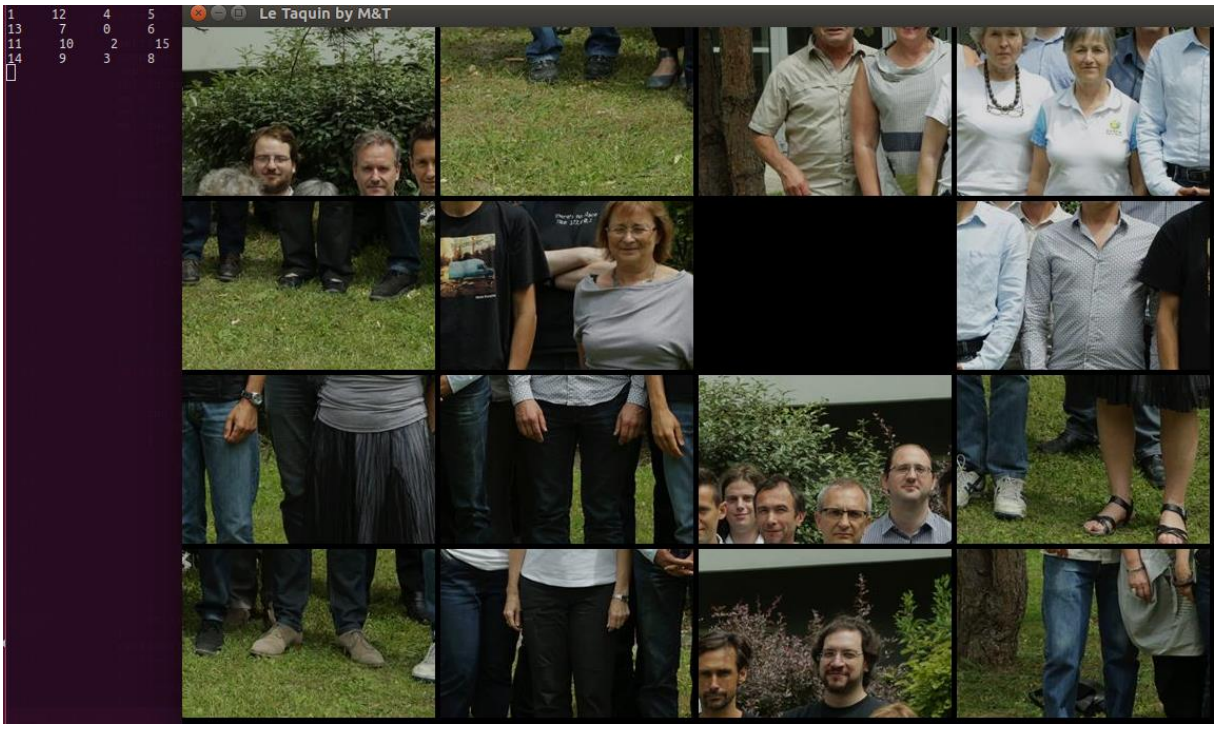
Pour montrer cela voici des captures d'écran. On peut voir que la valeur de 0 correspondra toujours à la case noire.



ici on peut voir que le 1 et le 0 se sont inversé



Et en considérant le mélange :



Et c'est là que entre en jeu la fonction « *verification* » pour « vérifier » si toutes les valeurs sont dans l'ordre ou non, si non, le jeu continue.

V – Le mélange

Le mélange dans notre cas, n'est pas vraiment un mélange. On pourra appeler cela une « escroquerie » je dirai même. L'algorithme, consiste ici à prendre une valeur aléatoire entre 0 et 3 (cela pouvait être 3300 et 3303, ça ne change rien, il faut juste 4 valeurs différentes) , et si la valeur est égale à 0, le contenu du tableau qui contenait 0 prend la valeur de l'entier de droite (dans le tableau). Mais il faut répondre à la condition que si 0 est déjà tout à droite, qu'il ne puisse pas de force se déplacer ce qui provoquera surement une erreur de segmentation. Si la valeur est 1, il se « déplace « en bas (il faut alors vérifier que si 0 est déjà tout en bas, qu'il ne se « déplace » pas quand même). Si la valeur est 2, elle se « déplace » à gauche (il faut alors vérifier que si 0 est déjà tout en gauche, qu'il ne se « déplace » pas quand même), et si la valeur est 3, le 0 se « déplace » en haut (il faut alors vérifier que si 0 est déjà tout en bas, qu'il ne se « déplace » pas quand même). Et cette action est répétée 2000 fois de telle sorte à prendre en compte le cas où rien ne se passe, et de telle sorte à obtenir un bon mélange.

En soit, ce n'est donc pas vraiment un mélange, cela revient à avoir l'image complète puis à changer de place la case noir ce qui va entrainer le déplacement de toutes les autres cases et à grande échelle, cela provoque un « mélange ».

VI – Conclusion

Majdi :

Ce projet pour moi a été pour la première fois un vrai challenge. Dans toute ma scolarité je m'étais dit que le travail de groupe ne pouvait que me freiner dans mon apprentissage et c'est aujourd'hui que j'apprends que j'ai eu tort (oui ça a pris du temps ^^') et surtout dans un domaine comme l'informatique. Je serai ravi de participer de nouveau à des projets. C'était une bonne expérience.

Thomas :

J'ai bien apprécié ce format d'apprentissage qui nous a permis d'évoluer en autonomie mais avec un minimum de critères à respecter. Le travail était selon moi une bonne expérience. La communication était primordiale.