GUERCI Richard *Groupe 1*
FERRARI Nicolas

Projet Assembleur 8086

I/ Mode d'emploi.

Le jeu est très simple d'utilisation.

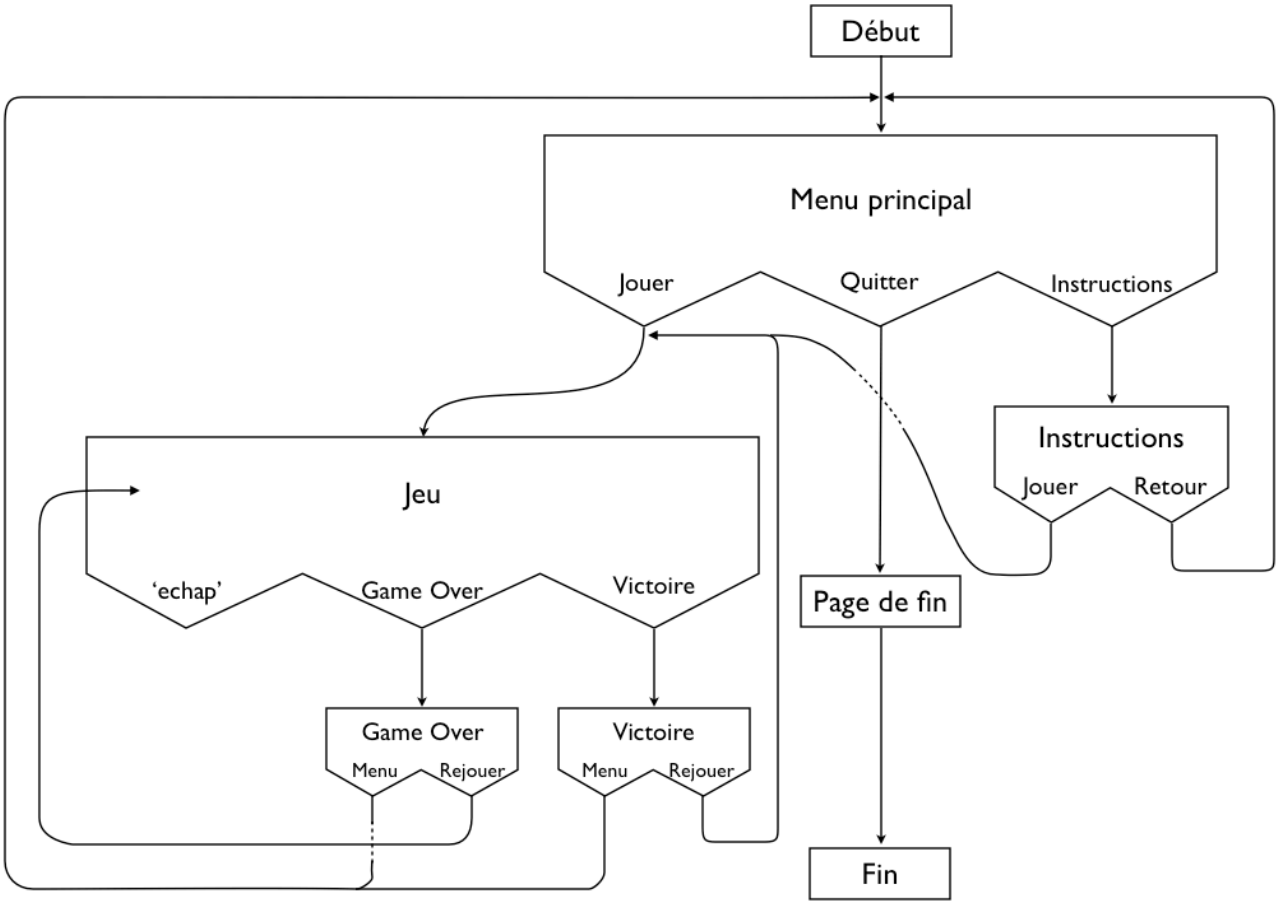
Commencez par dézipper l'archive 'guerci_ferrari.zip'.Allez dans le dossier 'laby' et double clique sur l'exécutable 'laby.com'. Le programme se lance.

Vous accédez alors au menu principal du jeu, sélectionnez votre choix grâce au curseur de la souris (en forme de clé).
Lorsque vous jouez, utilisez les touches z,q,s et d pour vous déplacer dans le labyrinthe.
Récupérez les clés pour passer les portes et accéder aux niveaux supérieurs et récupérez l'épée pour tuer les monstres.
Vous pouvez quitter a tout moment le jeu en appuyant sur 'echap'.

Quittez le programme en sélectionnant 'Quitter' dans le menu principal.

Code de triche: Appuyez sur 'n' puis sur 'espace' à n'importe quels moments du jeu pour passer un niveau (trop long ou trop difficile).

II/ Fonctionnement global du programme.



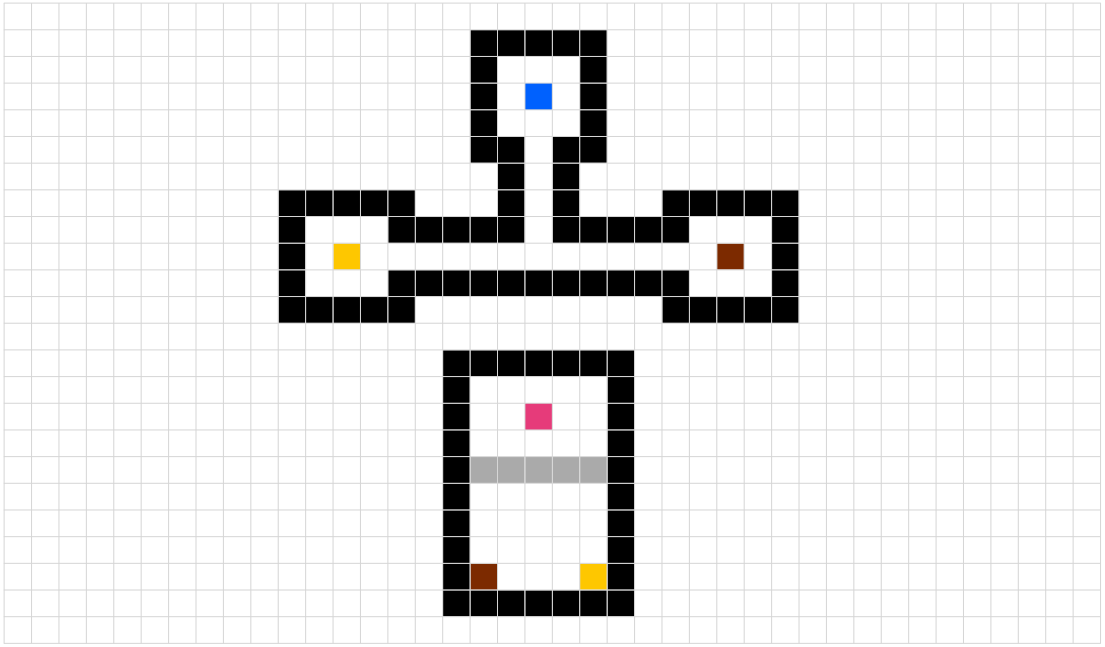
GUERCI Richard *Groupe 1*
FERRARI Nicolas

II/ Affichage du labyrinthe.

A/ Codage du labyrinthe.

Les labyrinthes sont codés sous forme de matrices de la manière suivante:

1. On dessine le labyrinthe sur une grille de 40x24 :



2. Puis on traduis la grille selon la tableau de correspondance suivant:

	ELEMENT	CODE
	Rien	0
	Mur	1
	Porte	2
	Clé	3
	Aventurier (face)	4
	Aventurier (dos)	5
	Aventurier (gauche)	6
	Aventurier (droite)	7
	Princesse	8
	Prison	9
	Épée	10
	Monstre	11

GUERCI Richard *Groupe 1*
FERRARI Nicolas

3. Enfin, il ne reste plus qu'a transcrire la grille sous forme de variable. Ce qui donne:

```
L1      db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,4,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,1,0,1,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,1,1,1,0,1,1,1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,1,0,3,0,0,0,0,0,0,0,0,0,0,0,2,0,1,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,8,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,9,9,9,9,9,1,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,0,0,0,3,1,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

B/ Lecture de la matrice et affichage.

Pour afficher cette matrice, il suffit de récupérer son offset:

Code:

```
lea bx,L1
```

On charge la valeur pointé par l'offset:

Code:

```
mov dx,[bx]
```

On compare la valeur avec les valeur du tableau de correspondance (page 2) et on affiche le l'élément correspondant

Code:

```
        cmp dl,1h
        je motif_mur
        cmp dl,2h
        je motif_porte
        ...

motif_mur:
        lea bx,matrice_mur
        jmp affiche_motif
motif_porte:
        lea bx,matrice_porte
        jmp affiche_motif
        ...

affiche_motif:
        call aff_img
```

L'affichage des éléments sera explicité en II/ C/ et D/ page 5.

On recommence en incrémentant l'offset de 1 jusqu'à la fin de la matrice (offset + (40x24) - 1), on stockera cette valeur dans la variable 'limit':

Code:

```
inc bx

cmp bx,limit
jne debut
```

A noter: chaque motif fait 8x8 pixels, donc après chaque incrémentation de l'offset, l'abscisse (cx) est incrémenté de 8 (pour passer à la colonne suivante) jusqu'à la limite des abscisses (320: dimension de l'écran).
Lorsque cette limite est dépassé on incrémente l'ordonnée (dx) de 8 (pour passer à la ligne suivante) et on réinitialise l'abscisse.

Remarque: l'explication a été simplifié pour une meilleur compréhension.

L'affichage de la matrice des labyrinthes est géré par la procédure 'affichage_labyrinthe'

GUERCI Richard *Groupe 1*
FERRARI Nicolas

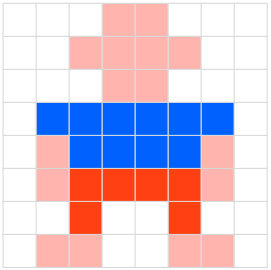
C/ Codage des éléments.

Les éléments sont codés de la même manière que les labyrinthes à la différence que chaque valeur ne correspond pas a un élément, mais à un pixel de couleur donné par le tableau de correspondance suivant:

	COULEUR	CODE MATRICE	CODE PALETTE
	Noir	0	0h
	Gris (mur)	1	F7h
	Gris (porte)	2	F8h
	Jaune	3	37h
	Rouge brique	4	15h
	Brun (porte&cheveux)	5	16h
	Couleur peau	6	A7h
	Bleu (vêtement)	7	FCh
	Rouge (vêtement)	8	F9h
	Rose (robe princesse)	9	87h
	Blanc (épée)	10	F6h
	Vert	11	29h

On obtiendra alors une variable du type: Ce qui donnera :

```
matrice_homme_face  db 0,0,0,6,6,0,0,0
                    db 0,0,6,6,6,6,0,0
                    db 0,0,0,6,6,0,0,0
                    db 0,7,7,7,7,7,7,0
                    db 0,6,7,7,7,7,6,0
                    db 0,6,8,8,8,8,6,0
                    db 0,0,8,0,0,8,0,0
                    db 0,6,6,0,0,6,6,0
```



D/ Lecture de la matrice et affichage.

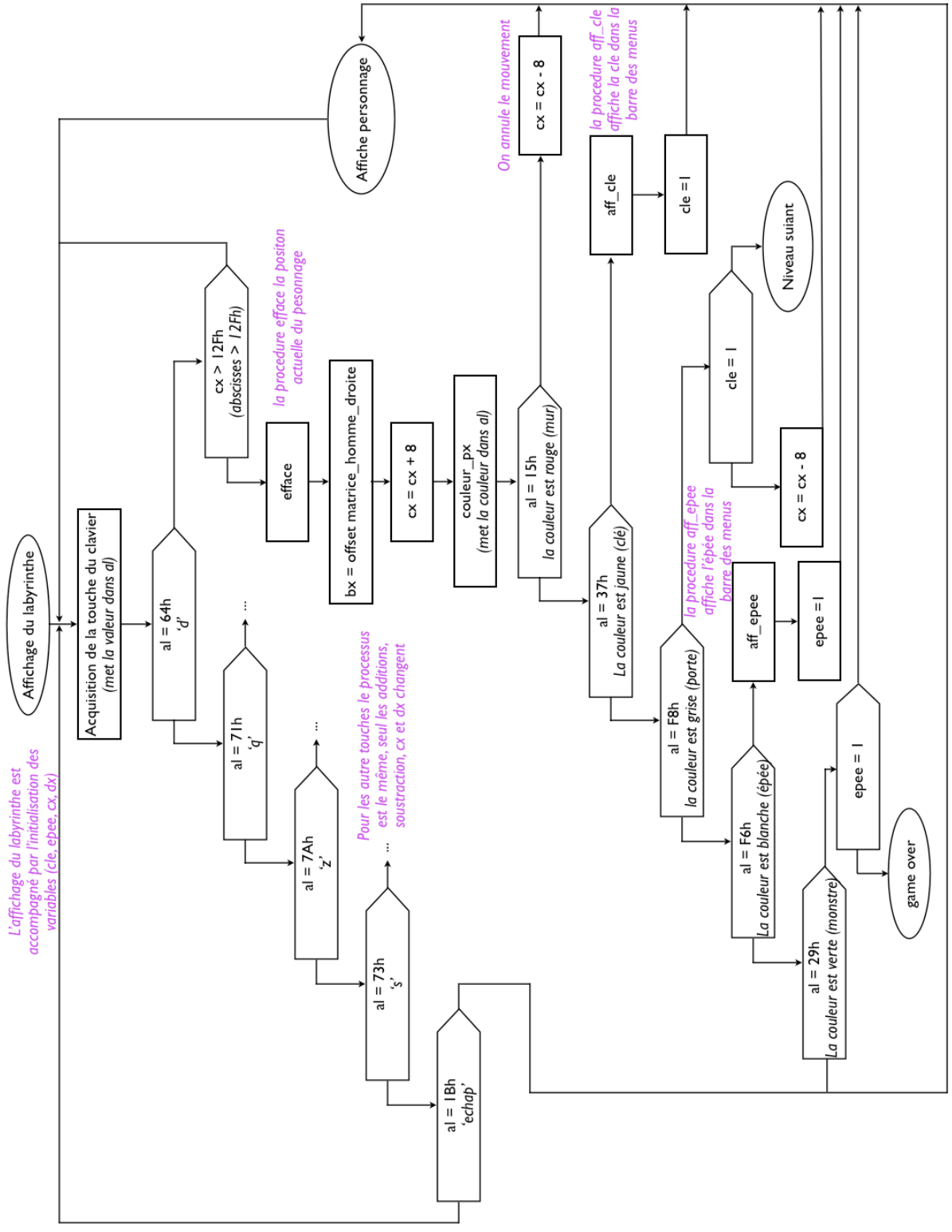
C'est la procédure 'aff_img' qui affiche les éléments. Elle marche exactement de la même façon que la procédure 'affiche_labyrinthe' à la différence que les valeurs de la matrice correspondent à des couleurs (selon le tableau ci-dessus). C'est-à-dire:
Code:

```
                cmp dl,0h
                je noir
                cmp dl,1h
                je gris_mur
                ...
noir:
                mov al,0h
                jmp aff_point
gris_mur:
                mov al,0f7h
                jmp aff_point
                ...
aff_point:
                mov ah,0Ch              ;écriture du pixel (la couleur dans al)
                int 10h
```

GUERCI Richard *Groupe 1*
FERRARI Nicolas

III/ Déplacements.

Voici un diagramme illustrant le principe de déplacement du personnage:



A noter: La couleur de référence est donné par la procédure ‘couleur_px’. Chaque éléments à une couleur bien défini en position 4x4 de la matrice. Ainsi si la couleur est rouge on sait que la prochaine position sera un mur, si c’est jaune une clé, ... Et on fait en conséquence: annulation du mouvement, game over, ...

GUERCI Richard *Groupe 1*
FERRARI Nicolas

IV/ Difficultés et améliorations.

Pas de difficulté particulière, la version finale ne présente aucun bug connu.
Il serai bien sûr possible d'améliorer le jeu en faisant bouger les monstres, en ajoutant des objets, en faisant un système de score, en rajoutant des niveaux, ou encore en générant des niveaux aléatoirement, ...

V/ Conclusion.

Le projet assembleur nous a aidé à mettre en pratique les logiques algorithmiques de programmation enseigné pendant le premier semestre.
Et il nous a permis de nous familiariser avec le fonctionnement d'un ordinateur (interruptions, mode graphique, stockage des données) avec un langage informatique de bas niveau.
Pour conclure, le sujet étant très intéressant, il nous a donné la motivation nécessaire pour développer une application performante, fiable et amusante.