

# BOBIMAC - AMINA FADILI & LAURINE SAJDAK

lien github du projet : <https://github.com/afadili/FLAPIMAC>

## I) CODE

### ORGANISATION DES FICHIERS

Les fichiers sont organisés en dossiers de headers (include), dossier de fichiers source (src), un dossier img qui contient toutes les textures nécessaires à l'affichage du jeu, un dossier obj pour les fichiers .o, puis un dossier ppm qui contient les fichiers ppm permettant le chargement des niveaux du jeu, et enfin un dossier sound pour les fichiers sons du jeu. Le dossier ttf n'est au final pas utilisé.

### COMPILEATION

On utilise un makefile qui permet de compiler directement tous les fichiers sources et créer ainsi un exécutable dans le dossier bin.

Pour compiler il faut faire un make, ensuite pour lancer le jeu il suffit de taper ./bin/flapimac dans le terminal.

A noter l'utilisation des extensions SDL : SDL\_image, SDL\_mixer (ainsi qu'un test de SDL\_ttf non fonctionnel)

## II) GAMEPLAY & FONCTIONNALITÉS

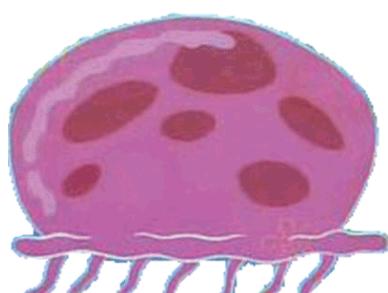
### LE JOUEUR



Le joueur incarne Bob l'éponge : Bob l'éponge se déplace horizontalement de manière automatique : sa valeur x s'incrémente à chaque tour de boucle (pour plus de détail, voir fichier main.c). Le joueur peut déplacer Bob vers le haut ou vers le bas à l'aide des flèches directionnelles du clavier. Il peut aussi lancer des projectiles avec la touche espace du clavier. L'amplitude de ce déplacement est déterminée par la variable speedMove\_y, initialisée à 1 dans le fichier display.c.  
Bob possède en théorie des points de vie : cependant, cette version du jeu s'arrête lorsque le joueur touche un ennemi ou un obstacle.

Bob est représenté par la couleur bleue dans le fichier ppm.

### LES ENNEMIS



Les ennemis sont représentés par des méduses. Les méduses se déplacent automatiquement et uniquement verticalement, à une vitesse donnée par la variable sppeMove\_y.

La direction de leur mouvement change si elles rencontrent un obstacle (player.c).

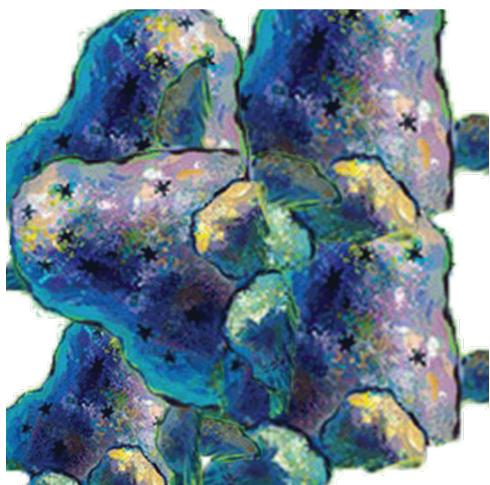
Bob meurt s'il touche une méduse.

Cette version du jeu n'inclut pas de lancer de projectiles fonctionnel.

Les ennemis meurent lorsqu'ils sont touchés par un projectile. Une mort augmente le score de 20 points.

Ils sont représentés par la couleur verte dans le fichier ppm.

## LES OBSTACLES



Les obstacles ne peuvent être franchis par aucune entité. Ils sont aussi redoutables que les ennemis puisqu'ils déclenchent un game over s'ils sont touchés par Bob. Cependant, Bob a la possibilité de détruire un obstacle : un obstacle possède en effet ici 3 vies. Pour enlever des vies à l'obstacle, Bob doit cependant posséder des points bonus, sinon, ses projectiles sont inefficaces et détruits lors d'une collision.

La destruction d'un obstacle permet d'augmenter le score du joueur, ici de 10 points.

Un obstacle est de couleur rouge dans le fichier ppm.

## LES BONUS

Pour faciliter sa victoire, Bob doit récupérer des bonus incarnés par un pâté de crabe.

Le nombre de bonus possédé par Bob est initialisé à 3 : chaque pâté de crabe augmente ce nombre de 3.

Un bonus récolté augmente ici le score de 5 points.

Bob ne peut détruire des obstacles que s'il possède des bonus en réserve : c'est pourquoi le joueur doit essayer de récupérer le plus possible de pâtes pour pouvoir se frayer un chemin dans la map, tuer plus d'ennemi et augmenter considérablement son score.

Un bonus est de couleur magenta dans le fichier ppm.

## LES PROJECTILES



Le nombre de projectiles de Bob est illimité : le joueur peut appuyer autant de fois qu'il le veut sur la touche espace.

Cependant, chaque tir enlève ici 0.5 point au score du joueur.

Ces projectiles peuvent tuer un ennemi ou un obstacle (à condition d'avoir des bonus en réserve).

Un projectile se détruit lorsqu'il touche un obstacle. Cependant, il tue les ennemis en les traversant : ainsi, on peut tuer plusieurs ennemis en un seul tir.

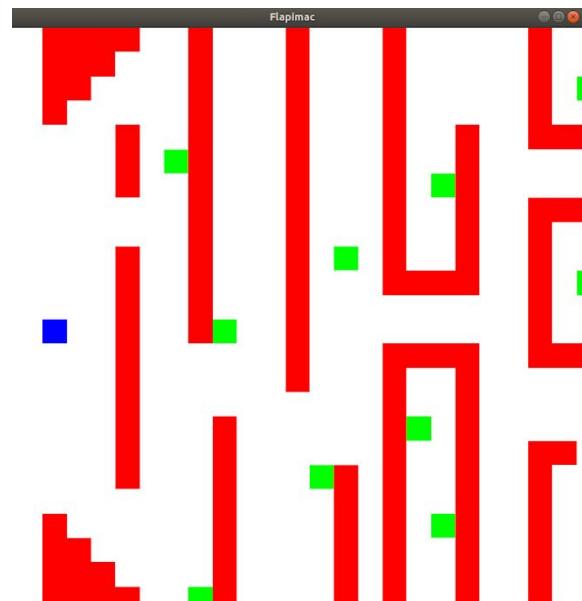
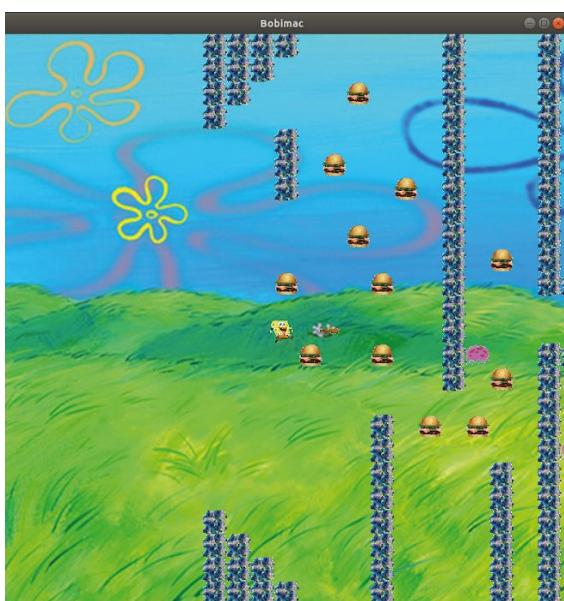
## LIGNE D'ARRIVÉE



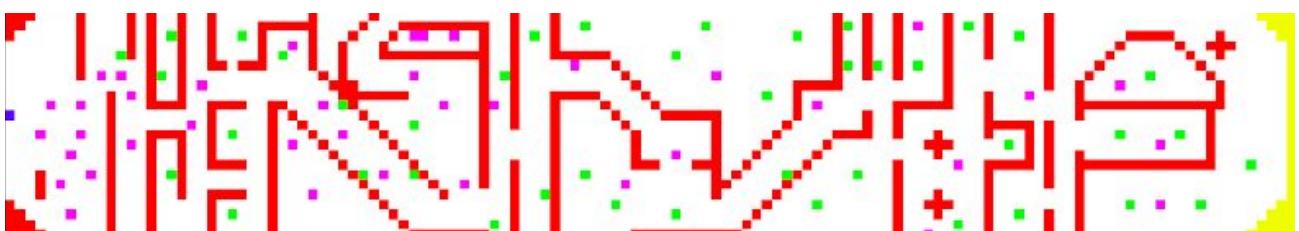
La partie est terminée lorsque Bob atteint la ligne d'arrivée représentée par sa maison ananas.

La ligne d'arrivée est représentée par la couleur **jaune** dans le fichier ppm.

## NIVEAU



Le niveau est chargé à partir d'un fichier ppm (voir partie gameplay pour les couleurs). Dans un premier temps, les blocs (1X1) de couleurs sont chargés à partir du fichier ppm (game.c), puis les textures sont ajoutées (display.c).



### **III) CHARGEMENT DU NIVEAU**

La gestion du niveau du jeu a été gérée à l'aide d'un fichier ppm, qui a permis de dessiner la map du jeu, en utilisant le logiciel Gimp, on a pu dessiner une carte du jeu, en mettant chacune des entités avec une couleur bien spécifique, pour pouvoir par la suite gérer chacune de ces entités dans le jeu.

La fonction qui gère le chargement du niveau se trouve dans notre game.c, cette fonction lit le fichier ppm et récupère les informations sur la largeur, la hauteur du fichier, ainsi que les couleurs de chaque pixel, et selon la couleur de chaque pixel récupérée, on alloue de la mémoire pour une entité bien déterminée selon cette couleur, ce qui nous permet de créer l'environnement du jeu selon ce fichier ppm.

### **IV) GESTION DES COLLISIONS**

Pour gérer les collisions, on passe en paramètre dans la fonction collision (entite.c) deux entités. On soustrait les valeurs x de l'entité 1 avec celle de l'entité deux, on prend la valeur absolu de ce résultat qu'on multiplie ensuite par deux étant donné que chaque entité est un carré de 1x1. On fait de même pour les valeurs y : si les deux résultats sont inférieurs à 2, c'est qu'il y a collision : on retourne alors 1. Ce résultat est ensuite traité dans la fonction checkCollision afin de faire une action spécifique au type de chaque entité.

### **V) CHARGEMENT DES TEXTURES**

Au début, on a utilisé une simple primitive, le dessin d'un carré pour afficher les entités du jeu, en les différenciant avec des couleurs différentes, et après avoir avancé dans les fonctionnalités du jeu, on a pu afficher chaque entité avec une texture différente.

La problématique qui s'est posée au début a été qu'il fallait charger toutes les textures avant de pouvoir les dessiner dans la boucle d'affichage, c'est pourquoi on a utilisé une fonction qui se trouve dans notre display.c qui commence par lire le contenu d'un dossier en utilisant la bibliothèque dirent.h de gestion des dossiers, ce qui a permis de mieux organiser nos textures.

Cette fonction a permis de charger toutes les textures du jeu et de les stocker dans un tableau préalablement déclaré comme variable globale, chaque texture va ensuite pouvoir être dessinée séparément avec un id différent.