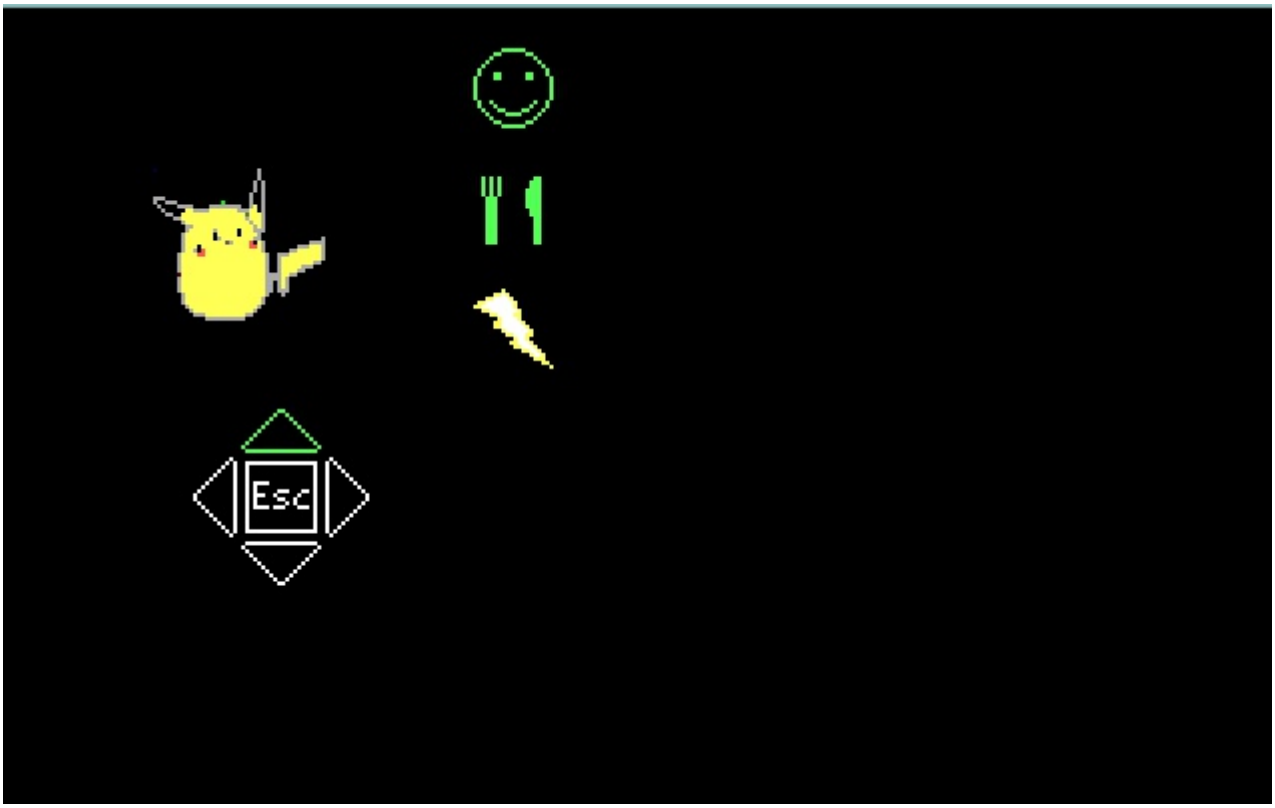
Pika-Goshi

LE JEU DE SIMULATION ELECTRIQUE !

Documentation Technique



Générateur aléatoire

L'aléatoire est basé sur les seules sources non prévisibles : le temps et les actions du joueur. Les centièmes de seconde de l'heure du système sont la source privilégiée.

```
mov ah, 2Ch ;récupère l'heure système
int 21h

test dl, 1 ;teste si le centième de seconde est impair
jz  play_with_the_pet_set_win_key_to_right
```

Ici les chances sont d'une sur deux, et l'heure à laquelle le temps système est requis dépend entièrement de l'action du joueur (ici lorsqu'il appuie sur "Jouer avec le familier").

Évolution du familier basée sur le traitement qu'il a eu

L'évolution est basée sur un arbre binaire, le familier peut à chaque évolution évoluer en bien ou en mal selon qu'il a été plus ou moins bien traité. Le système est simple, et se base sur les valeurs d'état du familier affichés à droite : si le nombre de fois où ces états sont en verts est supérieur au nombre de fois où ils sont en rouge alors le familier évolue en bien, dans le cas contraire il évolue en mal.
L'évolution a lieu après un durée fixée lors de l'évolution précédente.

Affichage de sprites

Il s'agit ici d'un gros morceau. Cela comprend toute la chaîne du travail des sprites sous un logiciel tel que Paint jusqu'à l'affichage du sprite à l'écran.
Il y a deux difficultés majeures, voici leur description et la solution apportée.

Quelle structure de données utiliser et comment l'afficher à l'écran ?

La structure de données utilisée est composée de la largeur et la hauteur du sprite, puis des pixels organisés par lignes. L'adresse du sprite considérée est celle du premier byte des pixels. Il est absolument nécessaire que les données largeur et hauteur soient situées avant les pixels, puisque celles-ci par l'intermédiaire d'une simple multiplication vont permettre de déterminer notamment la taille totale du tableau de pixels. Dans la procédure show_sprite les pixels sont alors affichés en partant du dernier en bas à droite jusqu'au premier en haut à gauche – de gauche à droite pour la procédure show_inverted_sprite. Cette procédure s'inspire de l'interruption affichant un pixel qu'elle encapsule, et prend en paramètres la position d'affichage x et y ainsi que l'adresse du sprite à afficher. Une procédure clear_sprite utile lors du déplacement d'un sprite fonctionne de façon identique, excepté que la couleur du pixel affiché sera toujours noir.

Comment travailler avec des formats habituels tels que le PNG qui dispose de millions de couleurs alors que le compilateur ne comprend qu'un code assembleur à 16 couleurs – assez imbuvable du reste -- qui n'a aucun rapport avec ce format ?

Écrire à la main chaque sprite est un suicide psychologique et une source d'erreurs difficiles à détecter. Le C++ vient à la rescousse : un utilitaire (SpriteMaker.exe) développé rapidement par nos soins utilisant la SFML permet de convertir une image sous un format connu et habituel en un fichier .txt qui contiendra la structure équivalente en code assembleur. Un petit script (gen.bat) permet de générer rapidement tous les .txt nécessaires au fonctionnement du jeu. Les .txt générés sont alors inclus dans le code avec l'instruction include tools/my_sprite.txt.