**#004**

La réalisation proposée consiste en un jeu de tir. Trois sprites sont utilisés. Le premier (000) désigne la position du joueur à l'écran sous la forme d'un vaisseau (16x16 pixels). Le second (002) et le troisième (018) représentent respectivement un projectile (missile) et un ennemi (extraterrestre).

➔ des extraits du **picodico** sont disponibles à la fin de ce document pour expliquer les termes : FOR, PAIRS, FLR et RND.


La programmation débute par l'introduction des trois fonctions usuelles : _INIT, _DRAW et _UPDATE.

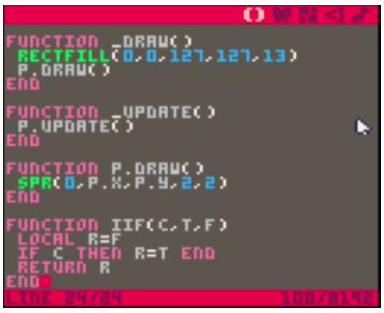

La variable (de type table) P reçoit l'ensemble des paramètres liés au joueur, en l'occurrence la position de son vaisseau à l'écran (P.X, P.Y). Cette variable est déclarée au début du programme afin d'être accessible par l'ensemble du programme. Les valeurs associées (X,Y) sont définies au sein de la fonction d'initialisation (_INIT).

La fonction _DRAW() accueille les traitements liés à l'affichage. Un rectangle plein gris-bleu (couleur n° 13) est dessiné sur l'ensemble de l'écran.


Les traitements spécifiques au joueur sont implémentés sous la forme de deux fonctions nommées P.DRAW et P.UPDATE. Ces fonctions sont respectivement appelées par _DRAW et _UPDATE.

La première (P.DRAW) dessine le vaisseau à l'écran à la position (P.X, P.Y).

La seconde (P.UPDATE) gère l'interaction avec le joueur.



Pour faciliter la gestion des conditions, un opérateur « ternaire » est créé sous la forme de la fonction IIF. Elle accepte trois arguments : C, T et F. La fonction renvoie T ou F selon que la condition est vraie (C==TRUE) ou non. L'écriture développée correspondant à `RESULT=IFF(C,T,F)` est :

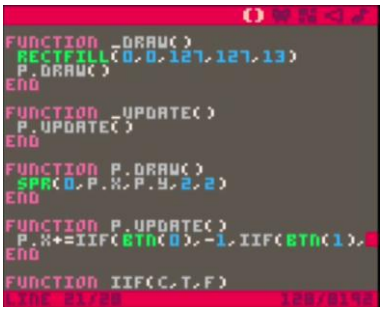
```
IF (C==TRUE) THEN
  RESULT=C
ELSE
  RESULT=F
END
```

L'usage de la fonction IIF permet de compacter la gestion des déplacements. Les 6 lignes ci-après :

```
IF BTN(0) THEN
 P.X-=1
END
IF BTN(1) THEN
 P.X+=1
END
```

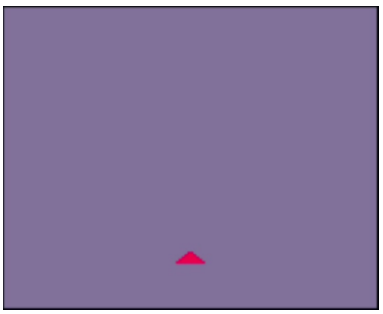
peuvent être résumées en une seule :

```
P.X=IIF(BTN(0),-1,IIF(BTN(1),1,0))
```

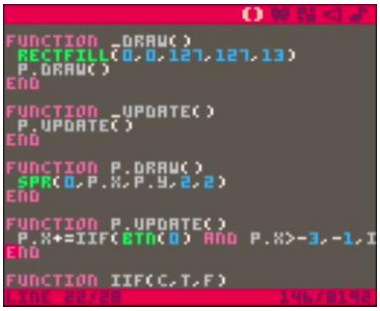


La gestion du déplacement du vaisseau à l'écran (effectuée dans P.UPDATE) en bénéficie.

L'expression, plus longue qu'à l'accoutumée, dépasse de l'écran. Le curseur permet de faire défiler la ligne à l'écran (pour la consulter et/ou la modifier).



Les prérequis étant en place, l'usage de la touche [Echap.] pour retourner à la console (>) suivi d'un « RUN » offre la possibilité d'essayer le programme.



```
FUNCTION _DRAW()
 RECTFILL(0,0,127,127,13)
 P.DRAW()
END

FUNCTION _UPDATE()
 P.UPDATE()
END

FUNCTION P.DRAW()
 SPR(0,P.X,P.Y,2,2)
END

FUNCTION P.UPDATE()
 P.X+=IIF(BTN(0) AND P.X>-3,-1,I
END

FUNCTION IIF(C,T,F)
```

En l'état, le vaisseau peut sortir par la gauche et la droite de l'écran. La valeur de P.X doit être restreinte à l'intervalle [-3,115].

D'où (en forme développée) :

```
IF BTN(0) AND P.X>-3 THEN
 P.X-=1
END
IF BTN(1) AND P.X<115 THEN
 P.X+=1
END
```

```
-1,IIF(BTN(1) AND P.X<115,1,0))
```

Soit (en forme compacte) :

```
P.X=IIF(BTN(0) AND P.X>-3,-1,IIF(BTN(1) AND P.X<115,1,0))
```

```
P={}
S={}
FUNCTION _INIT()
 P.X=56
 P.Y=96
 S.S={}
END

FUNCTION _DRAW()
 RECTFILL(0,0,127,127,13)
 P.DRAW()
END

FUNCTION _UPDATE()
 P.UPDATE()
END

FUNCTION P.DRAW()
```

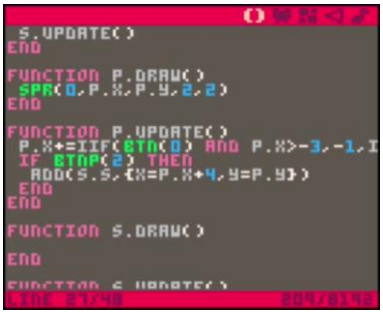
Les tirs du vaisseau sont gérés à l'aide de la variable (de type table) S qui comporte une variable (également de type table) S (soit S.S={}). Cette dernière liste l'ensemble des tirs effectués (et actifs). La déclaration (globale) de S s'effectue au début du programme. Son initialisation (ajout de S.S) intervient dans la fonction _INIT.

```
P={}
S={}
FUNCTION _INIT()
 P.X=56
 P.Y=96
 S.S={}
END

FUNCTION _DRAW()
 RECTFILL(0,0,127,127,13)
 P.DRAW()
 S.DRAW()
END

FUNCTION _UPDATE()
 P.UPDATE()
 S.UPDATE()
END
```

Le principe d'affichage et de gestion du vaisseau (P) est repris pour les tirs. Les fonctions S.DRAW et S.UPDATE sont respectivement appelées par _DRAW et _UDATE.

```
END

FUNCTION P.DRAW()
 SPR(0,P.X,P.Y,2,2)
END

FUNCTION P.UPDATE()
 P.X+=IIF(BTN(0) AND P.X>-3,-1,1
END

FUNCTION S.DRAW()
END

FUNCTION S.UPDATE()
END
```

Elles sont introduites en fin de programme.

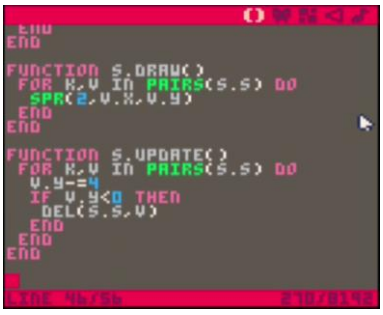


```
 S.UPDATE()
END

FUNCTION P.DRAW()
 SPR(0,P.X,P.Y,2,2)
END

FUNCTION P.UPDATE()
 P.X+=IIF(BTN(0) AND P.X>-3,-1,1
 IF BTNP(2) THEN
  ADD(S.S,{X=P.X+4,Y=P.Y})
 END
END

FUNCTION S.DRAW()

END
```

La fonction P.UPDATE (interaction avec le joueur pour piloter le vaisseau) évolue pour prendre en charge l'appui sur le bouton « flèche haut » (BTN(2)).

Lorsque ce bouton est pressé, un nouveau un tir est ajouté à la liste des tirs actifs (S.S).

Le tir est représenté par l'expression `{X=P.X+4, Y=P.Y}` où X et Y précisent sa position de départ (au centre à l'avant du vaisseau).

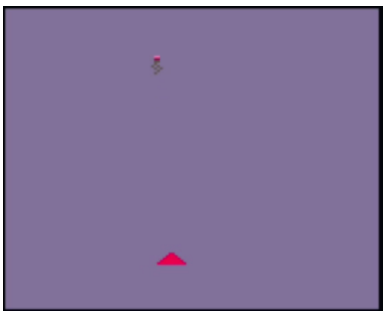


```
END
END

FUNCTION S.DRAW()
 FOR K,V IN PAIRS(S.S) DO
  SPR(2,V.X,V.Y)
 END
END

FUNCTION S.UPDATE()
 FOR K,V IN PAIRS(S.S) DO
  V.Y-=4
  IF V.Y<0 THEN
   DEL(S.S,V)
  END
 END
END
```

La fonction S.DRAW parcourt la liste des tirs actifs (à l'aide de la boucle FOR) et dessine tous les sprites représentant les projectiles à l'écran.

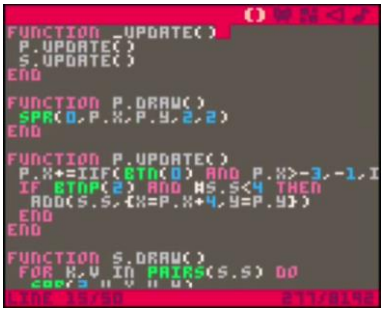
La fonction S.UPDATE parcourt la liste des tirs actifs (à l'aide de la boucle FOR) et déplace les projectiles vers le haut de l'écran (de 4 pixels). Lorsque le projectile sort de l'écran (V.Y<0), il est retiré de la liste (DEL).

➔ voir le **picodico** pour d'avantage d'explication sur **FOR** et **PAIRS**



[Echap.] suivi d'un « RUN » lance le programme. La touche « flèche haut » libère maintenant des projectiles qui traversent rapidement l'écran.

L'appui prolongé permet d'envoyer une multitude de projectiles.



Pour en limiter le nombre, la fonction P.UPDATE est modifiée. L'ajout d'un nouveau tir à la liste (S.S) est conditionné à l'usage du bouton 2 ET à la présence de moins de 4 tirs dans la liste (S.S).

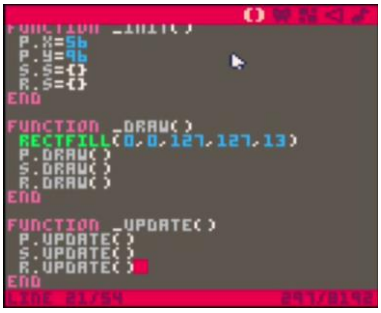
#S.S correspond au nombre de tirs dans S.S

```
IF BTN(2) AND #S.S<4 THEN
```

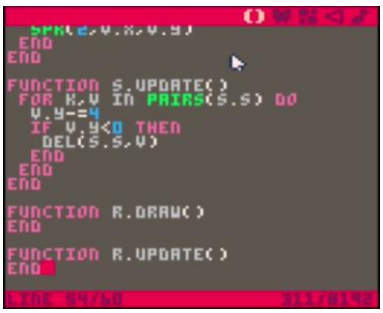
Tirer c'est bien… Mais tirer sur des ennemis, c'est mieux !

La variable R est introduite à cet effet. Elle comporte une liste d'ennemis (R.S={}).

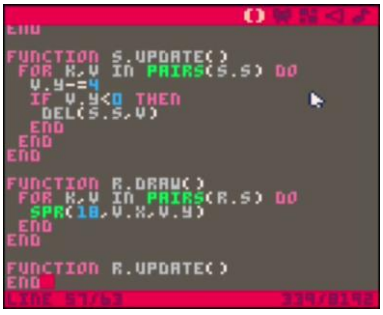
Le principe utilisé pour P (le vaisseau) et S (les tirs) est réutilisé pour R (les ennemis).



Les fonctions R.DRAW et R.UPDATE sont appelées par _DRAW et _UPDATE.

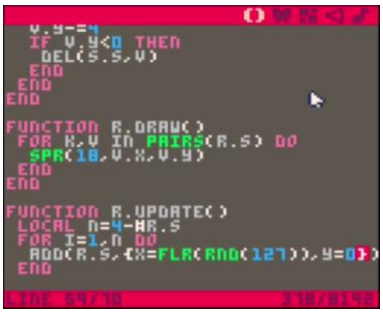


Les fonctions R.DRAW et R.UPDATE sont déclarées à la fin du programme.



La fonction R.DRAW exécute le même traitement que la fonction S.DRAW appliquée à la liste R.S et au sprite 18.

Pour mémoire, S.DRAW parcourt S.S et affiche le sprite 2.



La fonction R.UPDATE ajoute de nouveaux ennemis dans la liste R.S afin qu'elle en contienne toujours 4. Le nombre d'ennemis à ajouter (n) est donc 4-#R.S où #R.S correspond au nombre d'ennemis dans R.S.

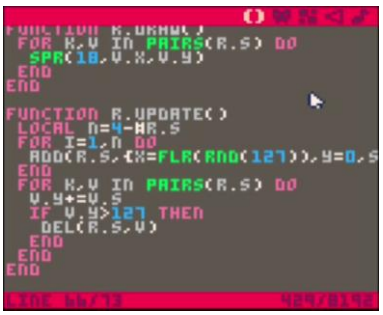
Les ennemis partent du haut de l'écran et se dirigent vers le bas en ligne droite. L'expression `{X=FLR(RND(127)),Y=0}` représente leur position de départ.

- `FLR(RND(127))` permet d'obtenir un nombre entier compris entre 0 et 126.

➔ voir le **picodico** pour davantage d'explications sur **FLR** et **RND**

L'expression est complétée par l'indication de la vitesse de déplacement (en pixels) : S.

- `FLR(RND(3))+1` permet d'obtenir un nombre entier compris entre 1 et 3.

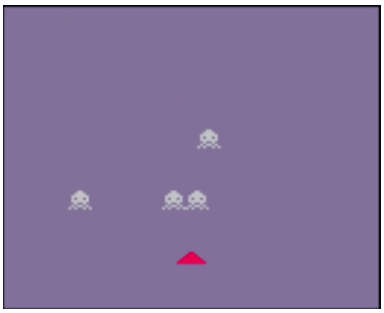


Suite à l'ajout des éventuels ennemis manquant, l'ensemble des ennemis (R.S) est parcouru pour les déplacer vers le bas de l'écran :

```
V.Y+=V.S.
```

Lorsque l'ennemi sort de l'écran, il est retiré de la liste :

```
IF V.Y>127 THEN
  DEL(R.S,V)
END
```



[Echap.], « RUN » permet de tester le comportement des ennemis. En l'état, les tirs effectués sont sans effet.

Les collisions entre les projectiles et les ennemis doivent être gérées afin du supprimer les uns et les autres.

La fonction _COLLISION() parcourt tous les projectiles. Pour chaque projectile, la fonction évalue la distance Manhattan* avec chaque ennemi. Lorsqu'elle est inférieure à 8, le projectile et l'ennemi sont retirés de leur liste respective :

```
DEL(S.S,V)
DEL(R.S,W)
```

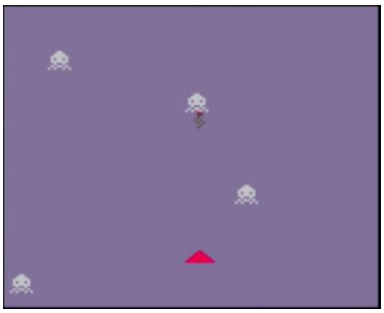
* voir https://fr.wikipedia.org/wiki/Distance_de_Manhattan

Le calcul de la distance Manhattan est effectué par :

```
ABS(V.X-W.X)+ABS(V.Y-W.Y)
```

```
END

FUNCTION _UPDATE()
 P.UPDATE()
 S.UPDATE()
 R.UPDATE()
 _COLLISION()
END

FUNCTION _COLLISION()
 FOR K,V IN PAIRS(S.S) DO
  FOR L,W IN PAIRS(R.S) DO
   IF ABS(V.X-W.X)+ABS(V.Y-W.Y)<
    DEL(S.S,V)
    DEL(R.S,W)
   END
  END
 END
END
LINE 24/88                    338/8192
```

La dernière action consiste à ajouter un appel à la fonction _COLLISION dans la fonction _UPDATE.



[Echap.], « RUN » lance le programme. Un tir (flèche haut) lorsque le vaisseau est face à un ennemi met en évidence la disparition du projectile et de l'ennemi (lors de le leur collision).

**PICODICO** (extrait)

FLR .....FLR renvoie l'entier le plus proche plus petit (ou égal).

```
FLR(-0.3) ➔ -1

FLR(1.3) ➔ 1
```

FOR .....FOR … DO … END est une structure de contrôle qui permet de répéter plusieurs fois un traitement.

```
FOR X=1,5 DO
Pour X=1 jusqu'à X=5 (en ajoutant 1 à chaque répétition) :
  PRINT(X)
  Affiche X
END
-- affiche 1,2,3,4,5
```

PAIRS .....PARIS est utilisé dans les boucles FOR pour parcourir une table (T) en fournissant à la fois la clef (K) et la valeur (V) de chaque élément. L'ordre des éléments parcourus n'est pas défini (et peut changer d'une fois à l'autre).

```
T = {["HELLO"]=3, [10]="BLAH"}
T.BLUE = 5
FOR K,V IN PAIRS(T) DO
  PRINT("K: " .. K .. " V:" .. V)
END

-- affiche
```

```
K: 10  v:BLAH
K: HELLO  v:3
K: BLUE  v:5
```

RND .....RND renvoie un nombre aléatoire.

`RND(X)` ➔ ? où 0 <= ? < X

Pour obtenir un entier, utiliser : `FLR(RND(X))`

Exemple : tirage d'un dé à 6 faces (1D6) :

```
PRINT(FLR(RND(6))+1)
```