

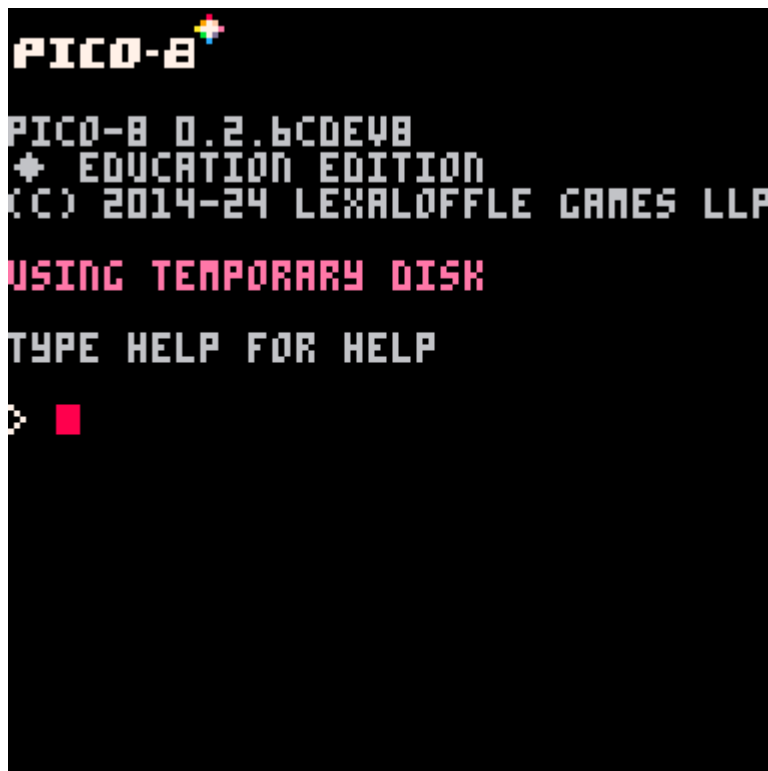
# Pico8

On va utiliser pico8. C'est une console de jeu imaginaire 8bit qui s'apparente à une vieille console comme la NES ou la Master System. On peut donc s'attendre à des caractéristiques terribles. Jugez plutôt :

- Une résolution de 128 pixel par 128
- Une palette de 16 couleurs affichable
- 4 channel sonore de 8 type de signaux
- 2 manette avec un croix directionnelle et 2 boutons

Normalement, Pico8 est un logiciel payant à installer. Mais il existe aussi une version éducative gratuite que l'on peut retrouver sur le site <https://www.pico-8-edu.com/>. La différence avec la version payante est que l'on ne peut pas enregistrer.

## Faisons un petit tour d'horizon de l'outil.



Voici l'écran au démarrage de la machine. On est accueilli par un écran donnant quelques informations dont la version de pico8 et un message indiquant que l'on peut afficher une aide en tapant la commande "HELP".

Il y a de quoi être perdu par rapport aux autres logiciels qui peuvent être bien plus accueillants avec des icônes à cliquer. Pico8 quant à lui est plutôt austère. Tu vois le petit > avec le carré rouge clignotant, c'est ce qu'on appelle un prompt. C'est dessus que l'on tape

des commandes. On essaie une commande ? (pas besoin d'appuyer sur MAJ ni de verrouiller les majuscules. Dans pico8 tout est en majuscule).

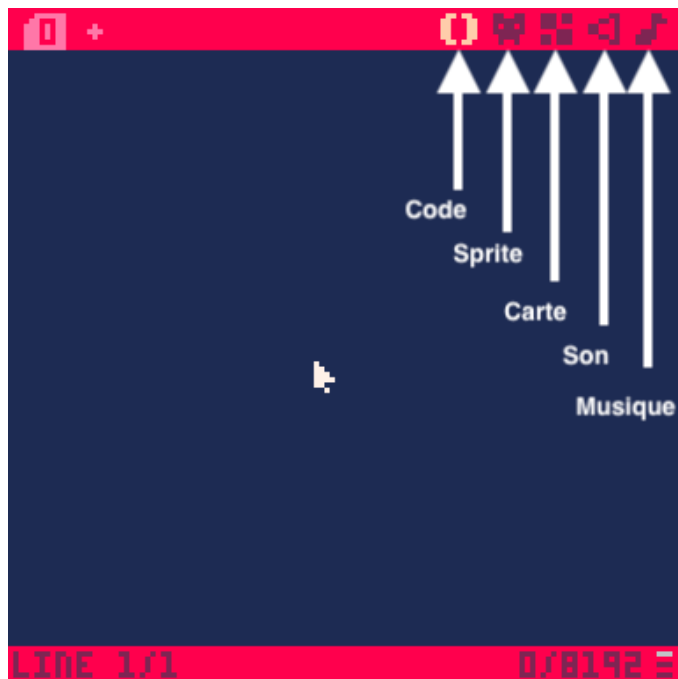
```
PRINT "BONJOUR"
```

Ensuite appuie sur Entrée pour exécuter la commande.



La machine te dit bonjour... Parce que tu lui as demandé de te dire bonjour.

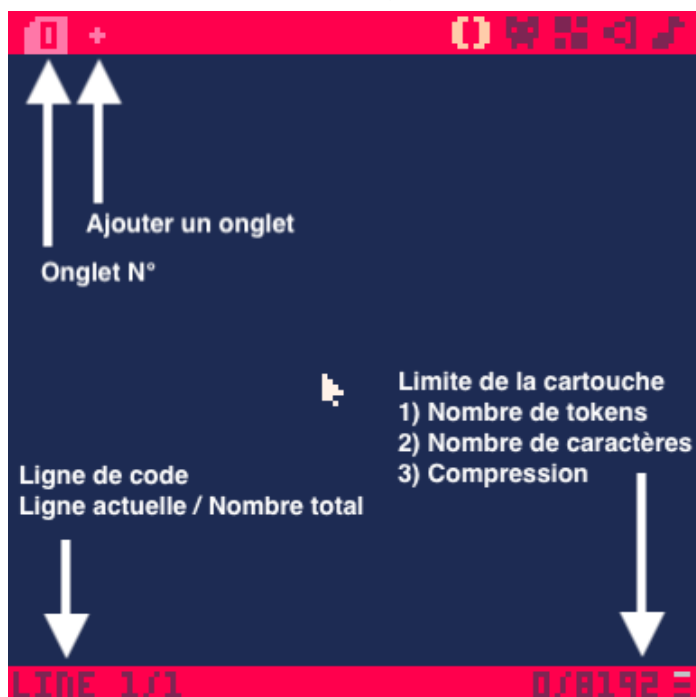
En appuyant sur la touche ESC, on bascule sur l'éditeur de code. C'est cette partie-là qui va nous intéresser. Il est composé de plusieurs écrans.



L'écran est petit, il n'y a pas beaucoup de place donc c'est très condensé. Un petit lexique s'impose.

- **Code** : C'est l'écran où l'on code notre programme. Il est en surbrillance parce qu'on est dans cet écran.
- **Sprite** : C'est l'écran où l'on dessine des petites sprites. Ce sont de petits dessins que l'on pourra animer ensuite.
- **Carte** : C'est l'écran où l'on dessine les cartes. On s'en sert principalement pour le décor.
- **Son** : C'est ici que l'on crée des sons pour notre jeu. Généralement des bruitages.
- **Musique** : Comme pour le son mais cette fois, c'est pour faire de la musique que l'on pourra jouer ensuite dans le jeu.

Dans le cadre de notre atelier, on va surtout coder. Donc on va rester sur l'écran de code.



# On va commencer par faire un petit programme

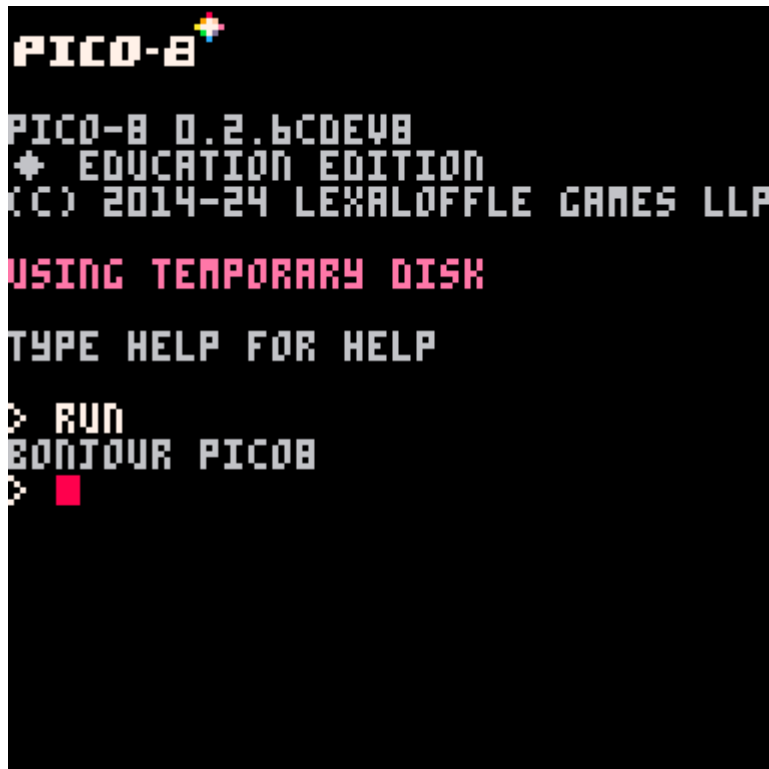
Tape ceci dans l'éditeur.

```
PRINT "BONJOUR PICO8"
```

Pour essayer ce petit programme. Appuie sur la touche ESC pour repasser sur l'écran de démarrage. Sur cet écran donc tape cette commande.

```
RUN
```

Et appuie sur Entrée pour exécuter le programme.

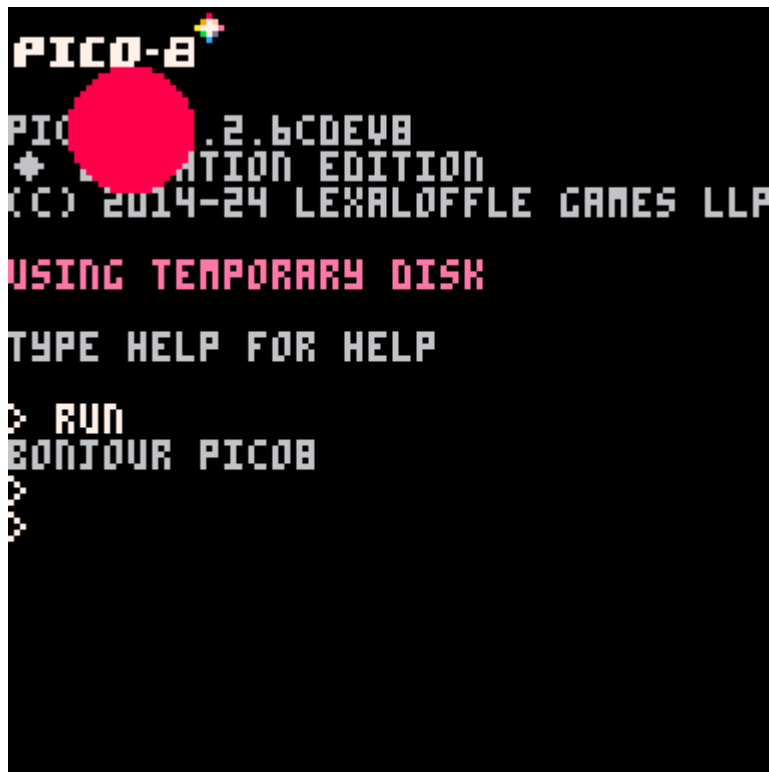


La machine dit bonjour pico8.

Mais on ne va pas se cantonner à écrire des messages, on peut dessiner aussi. Un cercle par exemple. Appui sur la touche ESC pour retourner sur l'éditeur de code et remplace tout le code par ceci.

```
CIRCFILL(20, 20, 10, 8)
```

On pourrait de nouveau tester en repassant sur l'écran principal (en appuyant sur ESC) puis en lançant la commande `RUN`. Mais il y a plus rapide et facile. Maintient la touche CTRL enfoncé et appui sur la touche R. C'est un raccourci que l'on va appeler CTRL+R.



On est revenu sur l'écran principal mais il y a un rond rouge maintenant au milieu du texte. C'est que pico8 exécute bêtement ce qu'on lui demande. Tu veux un rond rouge, tu auras un rond rouge. Et c'est ce que fait la fonction `CIRCFILL`. C'est une fonction qui dessine un cercle plein.

Mais si on ne veut que le rond rouge, on fait comment ? On pourrait lui demander d'effacer l'écran. Il existe une fonction pour cela qui s'appelle `CLS`. (Pourquoi `CLS` ? C'est la contraction de `CLear Screen` ce qui signifie "Nettoyer l'Écran"). Appuie sur ESC pour retourner sur l'éditeur (je demanderai juste de retourner sur l'éditeur les prochaines fois, tu sais comment faire maintenant). Insert ça en première ligne. (Je surligne en vert ce qu'il faut ajouter ;) )

```
CLS()  
CIRCFILL(20, 20, 10, 8)
```

Et utilise le raccourci CTRL+R pour voir le résultat.



Maintenant, on n'a que le rond rouge.

Mais il y a aussi le petit > avec le carré rouge clignotant, tu sais le prompt. C'est parce que le programme est terminé, il te rend la main.

Comment faire en sorte que le programme ne s'arrête pas ? Il y a une fonction qui est appelée par pico8 durant l'exécution du programme. Cette fonction est appelée tout le temps, en continu. Elle s'appelle `_DRAW` et c'est à toi de la créer comme ceci.

```
FUNCTION _DRAW()  
  CLS()  
  CIRC_FILL(20, 20, 10, 8)  
END
```

Maintenant, utilise le raccourci CTRL+R pour voir ce que ça donne.



Et voilà, plus de prompt. D'ailleurs tu remarqueras que tu ne peux plus rien taper parce que le programme est en cours d'exécution. Si tu veux l'arrêter, il faut appuyer sur la touche ESC. Et si tu appuie une seconde fois, tu retourneras dans l'éditeur de code.

On l'a vu, `_DRAW` est une fonction spéciale que l'on reconnaît au `_` devant son nom et qui est appelée constamment par pico8 pour dessiner. Ça n'est pas la seule fonction, il y en a deux autres.

La première est `_INIT`, elle est appelée au lancement du programme. On s'en sert pour initialiser des paramètres. Par exemple, on pourrait faire ceci. (Je barre et surligne en rouge ce qu'il faut retirer. Et donc en vert ce qui va le remplacer ;) )

```
FUNCTION _INIT()  
  X=40  
  Y=50  
  RAYON=15  
  COULEUR=10  
END  
  
FUNCTION _DRAW()  
  CLS()  
  CIRC_FILL(20X, 20Y, 10RAYON, 8COULEUR)  
END
```

Un petit CTRL+R pour voir ?



Le rond est maintenant jaune, il s'est déplacé et est plus grand.

Ce que l'on a fait, c'est créer la fonction spécial `_INIT` qui est appelée au lancement du programme. Cette fonction crée des variables `X`, `Y`, `RAYON` et `COULEUR`. Le `=` permet d'affecter une valeur à une variable. Si cette variable n'existait pas, elle se serait créée toute seule, sinon ça lui aurait changé la valeur.

Plus bas, on retrouve ces variables dans les paramètres de la fonction `CIRCFILL`. C'était les nombres qui étaient entre parenthèses (`20, 20, 10, 8`). Cette fois si, on ne leur donne pas une valeur, on les utilise !

Du coup si tu te demandais ce que signifiaient ces nombres. Les deux premier servent à positionner le cercle dans l'écran, le troisième à définir sa taille et le dernier à le colorer. Autant, les 3 premiers nombres se comptent en pixel, le dernier est un numéro de couleur dans une palette. La couleur 8 est un rouge et 10 un jaune.

Essai donc de changer certaines valeurs pour voir quel effet ça fait.

On a vu la fonction `_DRAW` qui dessine et la fonction `_INIT` qui permet d'initialiser des valeurs, la troisième va nous servir à animer. C'est la fonction `_UPDATE` et, tout comme `_DRAW`, elle est appelée 30 fois par seconde. En fait, pico8 appelle successivement `_UPDATE` pour mettre à jour des informations puis `_DRAW` pour dessiner le résultat. Ainsi, on peut obtenir une animation.



Pour en revenir au rond jaune à présent. Si on veut animer un déplacement, il suffit de changer les valeurs de `X` et `Y` dans la fonction `_UPDATE`. Comme ceci.

```
FUNCTION _INIT()  
  X=40  
  Y=50  
  RAYON=15  
  COULEUR=10  
END  
  
FUNCTION _UPDATE()  
  X=X+1  
  Y=Y-1  
END  
  
FUNCTION _DRAW()  
  CLS()  
  CIRCFFILL(X, Y, RAYON, COULEUR)  
END
```

Qu'est ce qu'il se passe quand tu fais CTRL+R ? Le rond jaune s'en va. Au revoir...

Dans la fonction `_UPDATE`, on a fait `X=X+1` ce qui revient à récupérer la valeur de la variable `X`, ajouter `1` et affecter le résultat à la variable `X`. `=` ne sert pas à comparer mais à affecter ! `Y=Y-1`, c'est la même chose sauf qu'on retire `1` à la variable `Y`.

Mais si on retire `1` à `Y`, pourquoi le rond va vers le haut ? C'est à cause de la manière dont pico8 gère les coordonnées. La coordonnée 0,0 se trouve en haut à gauche. 127,127 quant à elle est tout en bas à droite. Si tu veux, tu peux essayer en dessinant un pixel à ces coordonnées avec la fonction `PSET`. Ajoute ceci dans la fonction `_DRAW`.

```
FUNCTION _DRAW()  
  CLS()  
  CIRCFFILL(X, Y, RAYON, COULEUR)  
  PSET(0, 0, 7)  
  PSET(0, 127, 11)  
  PSET(127, 0, 14)  
  PSET(127, 127, 15)  
END
```

Si tu fais le raccourci CTRL+R, tu verras qu'il y a maintenant un pixel dans chaque coin.

La fonction `PSET` sert à dessiner un pixel, les deux premiers paramètres servent aux coordonnées. La dernière à la couleur. Ici les couleurs `7`, `11`, `14` et `15` correspondent respectivement à du blanc, vert, pêche et rose

J'ai dit tout à l'heure que la fonction `_UPDATE` est appelée 30 fois par seconde. On peut faire en sorte qu'il soit appelé 60 fois par seconde ! Ça peut donner des animations très fluides. Pour cela, tu as juste à remplacer la fonction `_UPDATE` par `_UPDATE60`. Fais l'essai !

## Une première explosion


Bon c'est mignon tout cela, on a des ronds et des pixels mais rien de bien explosif. On va arranger tout ça ! Mais d'abord une petite mise au point. Pour faire les feux d'artifice, on va jouer avec des particules. Ce sont de tout petits éléments de la taille d'un pixel, parfois un tout petit peu plus gros. Et c'est avec ces particules qu'on peut créer des effets comme de la pluie, du feu, une explosion, etc... On appelle cela la physique des particules.

Comment est-ce que l'on va s'y prendre ? Pour démarrer, on va créer une variable qui va contenir toutes les particules de la scène. Où ça ? Là où l'on initialise tout !

```
FUNCTION _INIT()  
    PARTICULES={}  
  
    X=40  
    Y=50  
    RAYON=15  
    COULEUR=10  
END
```

Cette ligne revient donc à créer une variable appelée `PARTICULES` et sa valeur est juste `{}`. C'est une table vide. Une table est un contenant comme une boîte où l'on peut mettre plein de choses dedans. Comme des particules par exemple !

Pour cela, on va créer un onglet où l'on codera tout ce qui concerne les particules. Il faut cliquer sur le + en haut à gauche.



```
FUNCTION _INIT()
  PARTICULES = {}

  X=40
  Y=50
  RAYON=15
  COULEUR=10
END

FUNCTION _UPDATE()
  X=X+1
  Y=Y-1
END

FUNCTION _DRAW()
  CLS()
  CIRCFILL(X, Y, RAYON, COULEUR)
  PSET(0, 0, 7)
  PSET(0, 127, 11)
END
```

LINE 2/22 62/8192

Dans cet onglet, on va tout d'abord créer une fonction pour ajouter une nouvelle particule dans le registre de particules.

```
--PARTICULES
FUNCTION AJOUTER_PARTICULE(
  X,Y,R,
  DX,DY,
  GRAVITE,
  RETRECI,
  TRAINEE,
  C_TABLE,
  DUREE)
LOCAL P={
  X=X,
  Y=Y,
  R=R,
  DX=DX,
  DY=DY,
  GRAVITE=GRAVITE,
  RETRECI=RETRECI,
  TRAINEE=TRAINEE,
  C=C_TABLE[1],
```

```

    C_TABLE=C_TABLE,
    T=0,
    DUREE=DUREE
}
ADD(PARTICULES, P)
RETURN P
END

```

C'est une grosse fonction, n'hésite pas à copier coller (CTRL+C puis CTRL+V) si tu ne le sens pas de tout recopier. Mais que fait elle ? Tout d'abord, elle a beaucoup de paramètre :

- **X** : va indiquer la position horizontale de la particule.
- **Y** : va indiquer la position verticale de la particule.
- **R** : va définir la taille de la particule. Si la valeur est 1, la particule ne fera qu'un pixel.
- **DX** : sert à indiquer la direction que prendra la particule horizontalement.
- **DY** : sert à indiquer la direction que prendra la particule verticalement.
- **GRAVITE** : va indiquer si la particule doit tomber vers le sol.
- **RETRECI** : va indiquer si la particule doit être de plus en plus petite.
- **TRAINEE** : sert à indiquer si la particule laisse une traînée lorsqu'elle se déplace.
- **C\_TABLE** : est une table qui va contenir toute les couleurs de la particule.
- **DUREE** : est le temps qu'il reste à la particule avant de disparaître.

Ensuite à l'intérieur de cette fonction, on crée une variable **P** qui est une autre table qui va contenir tout ce qui sera nécessaire à la particule. A savoir tout ce qui a été passé en paramètre de la fonction. Aussi quand tu vois une ligne comme **X=X**, ça signifie que l'on affecte à une clé de table appelée **X** la valeur du paramètre **X**. Mais il y a deux autres éléments dans la table :

- **C** : est la couleur de la particule. Elle est initiée à **C\_TABLE[1]** ce qui correspond à la première couleur présente dans la table **C\_TABLE**.
- **T** : indique depuis combien de temps existe la particule. Elle est initiée à **0**.

Au fait, tu as peut-être remarqué qu'il y a le mot **LOCAL** devant la déclaration de la variable **P**. Ça sert à indiquer que la variable n'existera que dans la fonction. Mais ne t'inquiète pas, tout ce que l'on a mis dans la variable ne va pas disparaître parce que, juste après la déclaration de cette variable, on appelle une fonction **ADD(PARTICULES, P)**. Cette fonction va ajouter dans une table (ici **PARTICULES**) une nouvelle valeur : la particule **P** que l'on a créé juste au-dessus.

Ensuite on a une ligne **RETURN P**. Cette commande indique que la fonction répond une valeur lorsqu'on l'appelle. Ici c'est la particule qui vient tout juste d'être créée.

Enfin un dernier petit détail, est-ce que as vu la première ligne avec `-- PARTICULES`, Tout ce qui après `--` n'est pas interprété, c'est un commentaire. Mais celui-ci est un peu spécial parce qu'il est en première ligne. Dans ce cas, il donne un nom à l'onglet. Si tu mets le curseur sur l'onglet, tu verras son nom apparaître.



```
--PARTICULES
PARTICULES = 11

X=40
Y=50
RAYON=15
COULEUR=10
END

FUNCTION _UPDATE()
  X=X+1
  Y=Y-1
END

FUNCTION _DRAW()
  CLS()
  CIRCFFILL(X, Y, RAYON, COULEUR)
  PSET(0, 0, 7)
  PSET(0, 127, 11)
END
```

LINE 2/22 128/8192

Maintenant que l'on a une table de particule et que l'on peut y ajouter des particules. Commençons à l'utiliser pour faire péter des trucs ! On va créer une fonction explosion ! Mais pour ça, on va créer encore un autre onglet. Clique sur le + en haut à gauche et tu pourras y mettre ce bout de code.

```
--EXPLOSION
FUNCTION EXPLOSION(
  X,Y,
  C_TABLE,
  NB,
  TRAINEE)
FOR I=0, NB DO
  AJOUTER_PARTICULE(
    X,          -- X
    Y,          -- Y
    3,          -- R
    RND(2)-1,   -- DX
    RND(2)-1,   -- DY
```

```

    TRUE,          -- GRAVITE
    TRUE,          -- RETRECI
    TRAINEE,       -- TRAINEE
    C_TABLE,       -- C_TABLE
    40+RND(25)     -- DUREE
)
END
END

```

Encore une grosse fonction à première vue mais c'est parce que celle-ci appelle la fonction qu'on avait fait plus haut. D'ailleurs tu remarqueras qu'elle utilise certains paramètres qu'on avait défini dans la fonction `AJOUTER_PARTICULE`.

- `X` : est la position horizontale où l'on va faire l'explosion.
- `Y` : est la position verticale où l'on va faire l'explosion.
- `C_TABLE` : est une table qui va contenir toutes les couleurs de cette explosion.
- `NB` : est le nombre de particules qu'il y aura dans cette explosion.
- `TRAINEE` : sert à indiquer si les particules de l'explosion vont laisser une traînée durant leur déplacement.

On a aussi ce qu'on appelle une boucle `FOR`. le `I=0`, `NB` signifie que cette boucle va compter de `0` jusqu'à la valeur de la variable `NB`. À chaque itération de cette boucle, on va appeler la fonction `AJOUTER_PARTICULE`. D'ailleurs cette fonction utilise beaucoup de paramètres (10), tu as dû remarquer qu'à la fin de chaque ligne de paramètre, il y a un commentaire. Ça n'a rien d'indispensable mais ça aide beaucoup à comprendre le code. Ça aide à savoir que parfois on réutilise un paramètre de la fonction `EXPLOSION` pour ensuite l'utiliser dans la fonction `AJOUTER_PARTICULE`. Et d'autres fois, on utilise des valeurs plus arbitraires.

Tu as peut-être remarqué une petite fonction appelée `RND`. C'est une fonction qui choisit un nombre aléatoire comme si on lançait un dé. Le paramètre de la fonction revient à choisir la taille du dé. Par exemple avec la fonction `RND(25)`, on va obtenir un nombre entre `0` et `25` mais on aura une infinité de possibilités : 1.2952, 21.234009, 13.034 et j'en passe.

Ok donc on va pouvoir faire des explosions. Alors aller ! BOOM ! ... Mais il faut pouvoir faire BOOM. On va donc faire en sorte qu'en appuyant sur un bouton, on ait une explosion. En temps normal, c'est la fonction `_UPDATE` qui se charge des interactions. Alors tu vas faire ces modifications. On va aussi en profiter pour faire un peu de ménage.

```

FUNCTION _INIT()
    PARTICULES={}

```

```

X=40
Y=50
RAYON=15
COULEUR=10
END

FUNCTION _UPDATE()
X=X+1
Y=Y-1

--LANCER UN FEU D'ARTIFICE
IF BTNP(X) THEN
EXPLOSION(
    RND(128),      -- X
    RND(128),      -- Y
    {10, 9},       -- C_TABLE
    20,            -- NB
    TRUE           -- TRAINEE
)
END
END

FUNCTION _DRAW()
CLS()

CIRCLEFILL(X, Y, RAYON, COULEUR)
PSET(0, 0, 7)
PSET(0, 127, 11)
PSET(127, 0, 14)
PSET(127, 127, 15)
END

```

**IF** est une instruction qui va exécuter du code selon la condition qui est écrite juste à sa droite. Dans notre cas, c'est écrit **BTNP(X)**. Cette fonction va indiquer si l'on a appuyé sur le bouton **X** de la manette. Le pico8 est une console dont la manette dispose d'une croix directionnelle et des boutons et **X** et **O**. Mais je ne peux pas te la montrer vu que la console n'existe pas.

On essaie ? Tu te rappelles du raccourci pour lancer le jeu ? C'est CTRL + R ! Et maintenant appuie sur le bouton **X** de la manette ... C'est la touche X du clavier. Il se passe rien ? Hum oui quand tu appuie sur le bouton, tu déclenche une explosion qui va ajouter des particules dans la table. Mais il faut dessiner ces particules ! On va créer une

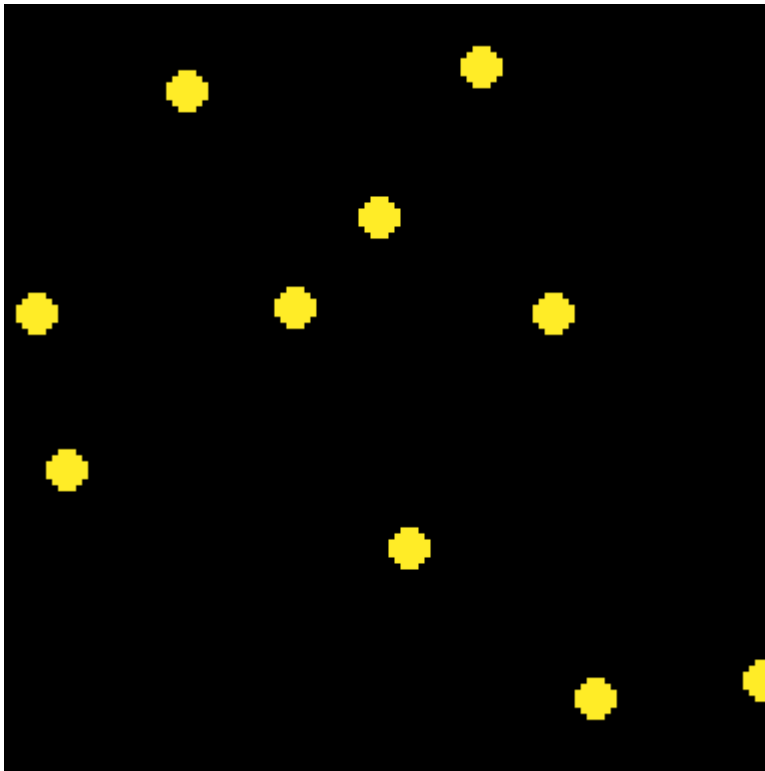
nouvelle fonction `DRAW_PARTICULES` dans l'onglet marqué Particules, tu peux voir le nom des onglets en mettant le curseur dessus.

```
FUNCTION DRAW_PARTICULES()  
  FOR P IN ALL(PARTICULES) DO  
    CIRCFFILL(P.X,P.Y,P.R,P.C)  
  END  
END
```

Ensuite retourne dans l'onglet principal (le numéro 1) et ajoute ceci dans la fonction `_DRAW`

```
FUNCTION _DRAW()  
  CLS()  
  
  DRAW_PARTICULES()  
END
```

Maintenant essaie pour voir !



Alors si tu as l'impression que tu n'as pas des explosions mais que l'écran a l'air de ressembler de plus en plus à une nuit étoilée... mais avec de grosses étoiles... c'est normal.



Parce que ce que l'on a fait pour le moment, c'est d'afficher un rond là où se trouve chaque particules. On avait vu que la boucle `FOR I=0, NB DO` signifie que cette boucle va compter de 0 jusqu'à la valeur de la variable NB. `FOR P IN ALL(PARTICULES) DO` va quant à lui passer en revue toutes les valeurs contenues dans la table `PARTICULES`. Et à chaque itération, chaque valeur sera dans la variable `P` créée pour l'occasion.

Du coup, ce que l'on demande de faire à chaque itération, c'est de dessiner un cercle plein avec la fonction `CIRCFILL`. Par contre, as-tu remarqué comment on passe les paramètres dans la fonction ? On utilise `P.X`, `P.Y`, `P.R` et `P.C`. Ça signifie que l'on récupère la valeur X, Y, R ou C de la variable `P`, la particule. Tu te rappelles ? Les particules qui ont été créées avec l'explosion.

Pour que l'explosion explose, il faut que ça bouge. Il faut modifier les particules entre chaque image affichée. On va maintenant créer une nouvelle fonction `UPDATE_PARTICULES` dans l'onglet marqué Particules, tu peux voir le nom des onglets en mettant le curseur dessus.

```
FUNCTION UPDATE_PARTICULES()  
  FOR P IN ALL(PARTICULES) DO  
    --DEPLACEMENT  
    P.X=P.X+P.DX  
    P.Y=P.Y+P.DY  
  END  
END
```

Ensuite retourne dans l'onglet principal (le numéro 1) et ajoute ceci dans la fonction `_UPDATE`

```
FUNCTION _UPDATE()  
  UPDATE_PARTICULES()  
  
  --LANCER UN FEU D'ARTIFICE  
  IF BTNP(☒) THEN  
    EXPLOSION(  
      RND(128),      -- X  
      RND(128),      -- Y  
      {10,9},        -- C_TABLE  
      20,             -- NB  
      TRUE            -- TRAINEE  
    )  
  END IF
```

```
END
END
```

Alors ça donne quoi cette fois ? Ça bouge ! Tout ça grâce à de petits  $P.X = P.X + P.DX$  et  $P.Y = P.Y + P.DY$ . Mais pourquoi ça part dans tous les sens ? Si tu te souviens quand on a créé la fonction `EXPLOSION`, on a passé des valeurs assez particulières à `DX` et `DY` : `RND(2) - 1`. Elle choisit un nombre aléatoirement entre 0 et 2. Mais comme on soustrait 1 ensuite, cela revient à avoir un nombre aléatoire entre -1 et 1. Faire ça sur le déplacement horizontal et vertical va donner l'impression que la particule part dans une direction imprévisible.

## Faire disparaître l'explosion

Mais on ne va pas s'arrêter là, tu as peut être remarqué que chaque particule continue son chemin jusqu'à sortir de l'écran. On pourrait le faire disparaître avant en utilisant les paramètres `DUREE` et `T` de la particule.

Pour commencer dans la fonction `UPDATE_PARTICULES` tu vas faire ces modifications.

```
FUNCTION UPDATE_PARTICULES()
  LOCAL MAJ_PARTICULES={}

  FOR P IN ALL(PARTICULES) DO
    --DUREE DE VIE
    P.T=P.T+1
    IF P.T<=P.DUREE THEN
      --DEPLACEMENT
      P.X=P.X+P.DX
      P.Y=P.Y+P.DY

      ADD(MAJ_PARTICULES,P)
    END
  END
  PARTICULES=MAJ_PARTICULES
END
```

Fait maintenant CTRL+R pour voir le résultat. Les particules disparaissent petit à petit. Encore une fois il y a un côté imprévisible dans la suppression des particules grâce à la valeur `40+RND(25)` passé dans le paramètre `DUREE`.

Et donc ce que l'on a changé dans la fonction `UPDATE_PARTICULES`, c'est que l'on ajoute 1 à l'attribut `T` de la particule et que si cette variable reste inférieure à la `DUREE`, on fait les mises à jour comme prévu. On a aussi créé une variable localement appelé `MAJ_PARTICULES`, Cette variable est une table de particule qui va être remplie des particules qui ne sont pas encore périmées. A la fin de la fonction, on remplace la variable `PARTICULES` par `MAJ_PARTICULES`. Ce qui revient à supprimer toutes les particules périmées.

Les particules disparaissent brusquement. Mais on a une variable `RETRECI`, on pourrait l'utiliser pour que la particule rétrécisse au point de disparaître complètement. Fais ces modifications dans `UPDATE_PARTICULES`.

```
FUNCTION UPDATE_PARTICULES()  
  LOCAL MAJ_PARTICULES={}  
  
  FOR P IN ALL(PARTICULES) DO  
    --DUREE DE VIE  
    P.T=P.T+1  
    IF P.T<=P.DUREE THEN  
      --PHYSIQUES  
      IF P.RETRECI THEN  
        P.R=P.R-0.1  
      END  
  
      --DEPLACEMENT  
      P.X=P.X+P.DX  
      P.Y=P.Y+P.DY  
  
      ADD(MAJ_PARTICULES,P)  
    END  
  END  
  PARTICULES=MAJ_PARTICULES  
END
```

Il va y avoir aussi une petite modification dans la fonction `DRAW_PARTICULES`.

```
FUNCTION DRAW_PARTICULES()  
  FOR P IN ALL(PARTICULES) DO  
    IF P.R<=1 THEN  
      PSET(P.X,P.Y,P.C)  
    END  
  END
```

```

ELSE
  CIRC_FILL(P.X,P.Y,P.R,P.C)
END
END
END

```

Essaie pour voir. C'est plus joli comme ça de voir la particule disparaître comme ça. D'ailleurs comme la particule rétrécit au point de ne faire qu'un pixel, ce n'est plus la peine de dessiner un cercle, autant dessiner un pixel.

## Ajoutons quelques petits trucs en plus

En plus de le rétrécir, on va aussi jouer avec la gravité et faire en sorte que les particules tombent. Fais cette petite modification dans la fonction `UPDATE_PARTICULES`.

```

FUNCTION UPDATE_PARTICULES()
  LOCAL MAJ_PARTICULES={}

  FOR P IN ALL(PARTICULES) DO
    --DUREE DE VIE
    P.T=P.T+1
    IF P.T<=P.DUREE THEN
      --PHYSIQUES
      IF P.GRAVITE THEN
        P.DY=P.DY+0.01
      END
      IF P.RETRECI THEN
        P.R=P.R-0.1
      END

      --DEPLACEMENT
      P.X=P.X+P.DX
      P.Y=P.Y+P.DY

      ADD(MAJ_PARTICULES,P)
    END
  END
  PARTICULES=MAJ_PARTICULES
END

```

Avec ces petites lignes, tu devrais voir les particules tomber lentement pendant leur déplacement. Essaie donc !

À présent, on va faire encore une petite évolution, on va jouer avec l'attribut `C_TABLE`. C'est une table de couleur pour rappel, on va l'utiliser de manière à ce que la couleur de la particule change pendant son déplacement. Il faut modifier la fonction `UPDATE_PARTICULES`.

```
FUNCTION UPDATE_PARTICULES()  
  LOCAL MAJ_PARTICULES={}  
  
  FOR P IN ALL(PARTICULES) DO  
    --DUREE DE VIE  
    P.T=P.T+1  
    IF P.T<=P.DUREE THEN  
      --LA COULEUR DEPEND  
      -- DE LA DUREE DE VIE  
      IF  
        P.T/P.DUREE<1/#P.C_TABLE  
      THEN  
        P.C=P.C_TABLE[1]  
      ELSEIF  
        P.T/P.DUREE<2/#P.C_TABLE  
      THEN  
        P.C=P.C_TABLE[2]  
      ELSEIF  
        P.T/P.DUREE<3/#P.C_TABLE  
      THEN  
        P.C=P.C_TABLE[3]  
      ELSE  
        P.C=P.C_TABLE[4]  
      END  
    END  
  
    --PHYSIQUES  
    IF P.GRAVITE THEN  
      P.DY=P.DY+0.01  
    END  
    IF P.RETRECIT THEN  
      P.R=P.R-0.1  
    END  
  END  
END
```

```

--DEPLACEMENT
P.X=P.X+P.DX
P.Y=P.Y+P.DY

ADD(MAJ_PARTICULES,P)
END
END
PARTICULES=MAJ_PARTICULES
END

```

Fais CTRL+R pour voir comment ça rend avec le changement de couleur. Bon là en vrai, ça passe juste du jaune à l'orange. Mais on va avoir une occasion d'alterné avec plus de couleur.

As-tu compris ce qu'il a été ajouté comme code ? On compare des ratios. D'un côté le ratio entre le temps (P.T) de la particule et sa durée de vie (P.DUREE). Et de l'autre, le ratio entre un nombre et #P.C\_TABLE. Le petit # devant P.C\_TABLE sert à connaître le nombre d'éléments qu'il y a dans une table. Dans chaque IF, le nombre qui sert de comparaison augmente. En fait l'idée derrière, c'est que, selon le nombre de couleur qu'il y a dans la table, qu'on passe successivement d'une couleur à l'autre le temps de la particule.

## Laisser une trace

La prochaine étape, va être de faire en sorte que la particule laisse une traînée lorsqu'elle se déplace. Ça va être plus long à faire mais ça vaut le coup.

Tu vas avoir besoin d'un nouvel onglet que l'on va appeler TRAINEE. Clique sur le petit + en haut à gauche et ajoute ceci.

```

--TRAINEE
FUNCTION TRAINEE(X,Y,L,
    C_TABLE,
    NB,
    DUREE)
FOR I=0, NB DO
    LOCAL NX=X+RND(L)-L/2
    LOCAL NY=Y+RND(L)-L/2
    --CONFIGURATION
    AJOUTER_PARTICULE(
        NX,          -- X
        NY,          -- Y

```

```

1,          -- R
0,          -- DX
0,          -- DY
FALSE,      -- GRAVITE
FALSE,      -- RETRECI
FALSE,      -- TRAINEE
C_TABLE,    -- C_TABLE
DUREE+RND(DUREE/2) -- DUREE
)
END
END

```

Ensuite il faut modifier ceci dans la méthode `UPDATE_PARTICULES`.

```

FUNCTION UPDATE_PARTICULES()
LOCAL MAJ_PARTICULES={}

FOR P IN ALL(PARTICULES) DO
--DUREE DE VIE
P.T=P.T+1
IF P.T<=P.DUREE THEN
--LA COULEUR DEPEND
-- DE LA DUREE DE VIE
IF
P.T/P.DUREE<1/#P.C_TABLE
THEN
P.C=P.C_TABLE[1]
ELSEIF
P.T/P.DUREE<2/#P.C_TABLE
THEN
P.C=P.C_TABLE[2]
ELSEIF
P.T/P.DUREE<3/#P.C_TABLE
THEN
P.C=P.C_TABLE[3]
ELSE
P.C=P.C_TABLE[4]
END

```

```

--PHYSIQUES
IF P.GRAVITE THEN
  P.DY=P.DY+0.01
END
IF P.RETREC I THEN
  P.R=P.R-0.1
END

--DEPLACEMENT
LOCAL OLD_X=P.X
LOCAL OLD_Y=P.Y
P.X=P.X+P.DX
P.Y=P.Y+P.DY

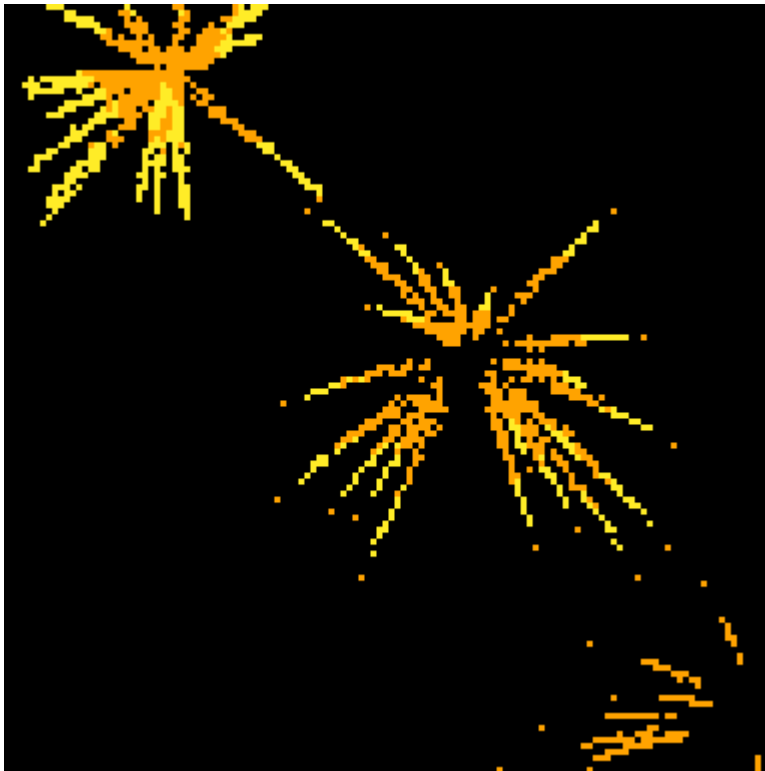
--TRAINEE
IF P.TRAINEE THEN
  TRAINEE(
    P.X, --X
    P.Y, --Y
    P.R, --L
    P.C_TABLE, --C_TABLE
    2, --NB
    20 --DUREE
  )
END

ADD(MAJ_PARTICULES,P)
END
END
PARTICULES=MAJ_PARTICULES
END

```

Tu vas pouvoir essayer à présent. C'est joli hein ? Ça ressemble déjà plus à un feu d'artifice maintenant !





Par contre tu as dû remarquer que ça rame de fou !!! Surtout quand il y a plusieurs explosions ! Avec la traînée on multiplie les particules au fur et à mesure qu'on fait évoluer la scène si bien qu'on en a rapidement des centaines. On va essayer de remédier à ça. Fais cette modification dans la fonction `TRAINEE`.

```
--TRAINEE
FUNCTION TRAINEE(X,Y,L,
    C_TABLE,
    NB,
    DUREE)
    LOCAL NOUVEAUX={}
    FOR I=0, NB DO
        LOCAL NX=X+RND(L)-L/2
        LOCAL NY=Y+RND(L)-L/2

        IF
            PAS_PRESENT(NOUEAUX,NX,NY)
        THEN
            --CONFIGURATION
            LOCAL P=AJOUTER_PARTICULE(
                NX,          -- X
                NY,          -- Y
```

```

1,          -- R
0,          -- DX
0,          -- DY
FALSE,      -- GRAVITE
FALSE,      -- RETRECI
FALSE,      -- TRAINEE
C_TABLE,    -- C_TABLE
DUREE+RND(DUREE/2) -- DUREE
)
ADD(NOUVEAUX,P)
END
END
END

FUNCTION PAS_PRESENT(TABLE,X,Y)
FOR P IN ALL(TABLE) DO
IF (FLR(P.X)==FLR(X)
AND FLR(P.Y)==FLR(Y))
THEN
RETURN FALSE
END
END
RETURN TRUE
END

```

Et ceci dans la fonction UPDATE\_PARTICULES.

```

FUNCTION UPDATE_PARTICULES()
LOCAL MAJ_PARTICULES={}

FOR P IN ALL(PARTICULES) DO
--DUREE DE VIE
P.T=P.T+1
IF P.T<=P.DUREE THEN
--LA COULEUR DEPEND
-- DE LA DUREE DE VIE
IF
P.T/P.DUREE<1/#P.C_TABLE
THEN

```

```

P.C=P.C_TABLE[1]
ELSEIF
P.T/P.DUREE<2/#P.C_TABLE
THEN
P.C=P.C_TABLE[2]
ELSEIF
P.T/P.DUREE<3/#P.C_TABLE
THEN
P.C=P.C_TABLE[3]
ELSE
P.C=P.C_TABLE[4]
END

--PHYSIQUES
IF P.GRAVITE THEN
P.DY=P.DY+0.01
END
IF P.RETRECI THEN
P.R=P.R-0.1
END

--DEPLACEMENT
LOCAL OLD_X=P.X
LOCAL OLD_Y=P.Y
P.X=P.X+P.DX
P.Y=P.Y+P.DY

--TRAINEE
IF P.TRAINEE
AND FLR(OLD_X)≠FLR(P.X)
AND FLR(OLD_Y)≠FLR(P.Y)
THEN
TRAINEE(
P.X,          --X
P.Y,          --Y
P.R,          --L
P.C_TABLE,    --C_TABLE
2,            --NB
20            --DUREE
)
END

```

```

    ADD(MAJ_PARTICULES, P)
END
END
PARTICULES=MAJ_PARTICULES
END

```

Si tu réessayes maintenant, ça devrait aller mieux. Ce que l'on a modifié consiste à ne pas ajouter de particule là où il y en a déjà.

## Et si on parlait d'une fusée

On a bien avancé, mais on peut encore ajouter des tas de trucs pour que ça ressemble de plus en plus à un feu d'artifice. On pourrait par exemple faire en sorte qu'une fusée parte du sol et éclate dans le ciel. On va faire ça toujours en jouant avec les particules ? C'est parti !

On va créer encore un nouvel onglet que l'on appellera FUSEE. Clique sur le petit + en haut à gauche et ajoute ceci.

```

-- FUSEE
FUNCTION LANCER_FUSEE(
    X_DEPART, Y_DEPART,
    X_CIBLE, Y_CIBLE,
    DUREE,
    COULEUR,
    TRAINEE)
LOCAL F={
    X=X_DEPART,
    Y=Y_DEPART,
    DX=(X_CIBLE-X_DEPART)/DUREE,
    DY=(Y_CIBLE-Y_DEPART)/DUREE,
    T=0,
    DUREE=DUREE,
    COULEUR=COULEUR,
    TRAINEE=TRAINEE
}
ADD(FUSEES, F)
END

-- METTRE A JOUR LES FUSEES
FUNCTION UPDATE_FUSEES()

```

```

LOCAL MAJ_FUSEES={}
FOR F IN ALL(FUSEES) DO
  --DUREE DE VIE
  F.T=F.T+1
  IF F.T<=F.DUREE THEN
    --DEPLACEMENT DE LA FUSEE
    F.X=F.X+F.DX
    F.Y=F.Y+F.DY

    TRAINEE(
      F.X,          --X
      F.Y,          --Y
      1,            --L
      {10,9,8,13}, --C_TABLE
      3,            --NB
      10            --DUREE
    )

    ADD(MAJ_FUSEES, F)
  END
END
FUSEES=MAJ_FUSEES
END

```

On va maintenant faire des petits ajustements dans les fonctions `_INIT` et `_UPDATE` pour que ce soit des fusées qui soient lancées. Il va falloir retourner dans l'onglet principal (le numéro 1)

```

FUNCTION _INIT()
  FUSEES={}
  PARTICULES={}
END

FUNCTION _UPDATE()
  UPDATE_FUSEES()
  UPDATE_PARTICULES()


  --LANCER UN FEU D'ARTIFICE
  IF BTNP(☒) THEN

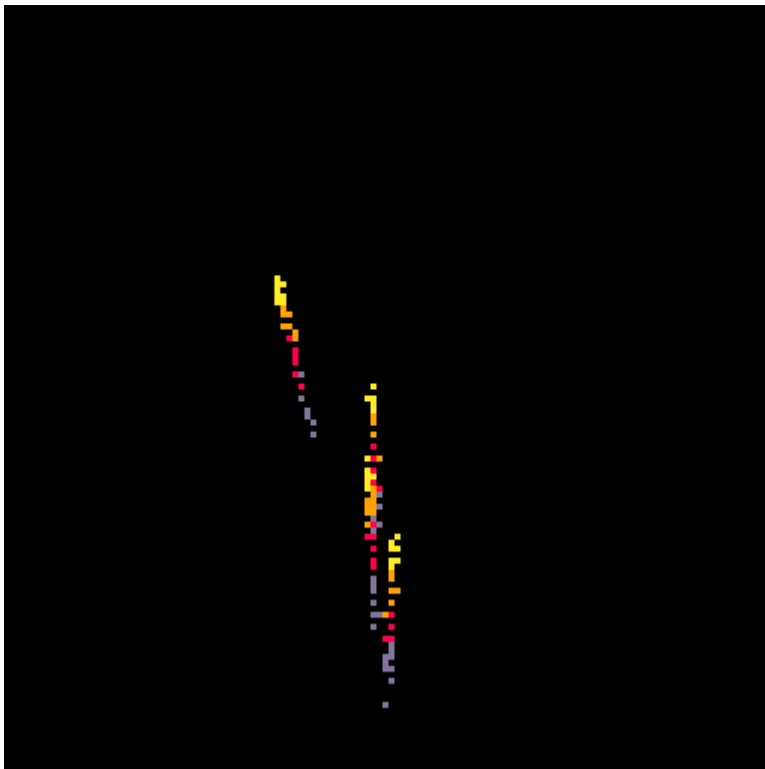
```

```

EXPLOSION(
    RND(128), -- X
    RND(128), -- Y
    {10,9}, -- C_TABLE
    20, -- NB
    TRUE -- TRAINEE
)
LANCER_FUSEE(
    63, -- X_DEPART
    127, -- Y_DEPART
    RND(127-40)+20, -- X_CIBLE
    RND(10)+15, -- Y_CIBLE
    50, -- DUREE
    {10,9}, -- COULEUR
    TRUE -- TRAINEE
)
END
END

```

Qu'est-ce que ça donne quand tu fais CTRL+R ? Si tu appuie sur le bouton , tu devrais voir une fusée partir.



Maintenant, il faut faire exploser cette fusée ! Modifie pour cela la fonction `UPDATE_FUSEES`.

```

FUNCTION UPDATE_FUSEES()
  LOCAL MAJ_FUSEES={}
  FOR F IN ALL(FUSEES) DO
    --DUREE DE VIE
    F.T=F.T+1
    IF F.T<=>F.DUREE THEN
      --EXPLOSION DE LA FUSEE
      EXPLOSION(
        F.X,      -- X
        F.Y,      -- Y
        F.COULEUR, -- C_TABLE
        20,       -- NB
        TRUE      -- TRAINEE
      )
    ELSE
      --DEPLACEMENT DE LA FUSEE
      F.X=F.X+F.DX
      F.Y=F.Y+F.DY

      TRAINEE(
        F.X,      --X
        F.Y,      --Y
        1,        --L
        {10,9,8,13}, --C_TABLE
        3,        --NB
        10        --DUREE
      )

      ADD(MAJ_FUSEES, F)
    END
  END
  FUSEES=MAJ_FUSEES
END

```

Voyons à quoi ça ressemble maintenant. Et boom !

Par contre, si tu appuie un peu trop frénétiquement sur le bouton pour envoyer les fusées, ça risque de se mettre à ramer. Il y a peut-être une astuce pour éviter cela, c'est de limiter le nombre de fusées dans l'écran. Il n'y a qu'à modifier les fonctions `_INIT` et `_UPDATE` pour cela.

```

FUNCTION _INIT()
  FUSEES={}
  FUSEES_MAX=3

  PARTICULES={}
END

FUNCTION _UPDATE()
  UPDATE_FUSEES()
  UPDATE_PARTICULES()

  --LANCER UN FEU D'ARTIFICE
  IF BTNP(✕)
    AND FUSEES_MAX>#FUSEES
  THEN
    LANCER_FUSEE(
      63,          -- X_DEPART
      127,         -- Y_DEPART
      RND(127-40)+20, -- X_CIBLE
      RND(10)+15,  -- Y_CIBLE
      50,          -- DUREE
      {10,9},      -- COULEUR
      TRUE         -- TRAINEE
    )
  END
END

```

Regarde le changement que ça fait. On ne peut plus envoyer autant de fusée qu'avant. Ça évite de trop ramer c'est toujours ça !

## Dernier détails à ajouter

Est-ce que ça te dit d'ajouter d'autres couleurs. Il faut juste quelques petites modifications dans les fonctions `_INIT` et `_UPDATE`.

```

FUNCTION _INIT()
  FUSEES={}
  FUSEES_MAX=4

```



```

COULEURS_EXPLOSION={
  {10,9},
  {11,3},
  {8,2},
  {12,1}
}

PARTICULES={}

END

FUNCTION _UPDATE()
  UPDATE_FUSEES()
  UPDATE_PARTICULES()

  --LANCER UN FEU D'ARTIFICE
  IF BTNP(☒)
    AND FUSEES_MAX>#FUSEES
  THEN
    LANCER_FUSEE(
      63,          -- X_DEPART
      127,         -- Y_DEPART
      RND(127-40)+20, -- X_CIBLE
      RND(10)+15,   -- Y_CIBLE
      50,          -- DUREE
      {10,9}        -- COULEUR
    COULEURS_EXPLOSION[
      FLR(RND(4))+1
    ],             -- COULEUR
    TRUE           -- TRAINEE
  )
END
END

```

C'est joli toutes ces couleurs, non ? Mais ce qui serait encore mieux, ce serait d'ajouter un flash au moment de l'explosion !

On va avoir besoin de faire quelques modifications dans la fonction `UPDATE_FUSEES`.

```

FUNCTION UPDATE_FUSEES()
  FUSEE_EXPLOSE=FALSE

```

```

LOCAL MAJ_FUSEES={}
FOR F IN ALL(FUSEES) DO
  --DUREE DE VIE
  F.T=F.T+1
  IF F.T>F.DUREE THEN
    --EXPLOSION DE LA FUSEE
    FUSEE_EXPLOSE=TRUE
    EXPLOSION(
      F.X,      -- X
      F.X,      -- Y
      F.COULEUR, -- C_TABLE
      20,       -- NB
      TRUE      -- TRAINEE
    )
  ELSE
    --DEPLACEMENT DE LA FUSEE
    F.X=F.X+F.DX
    F.Y=F.Y+F.DY

    TRAINEE(
      F.X,      --X
      F.Y,      --Y
      1,        --L
      {10,9,8,13}, --C_TABLE
      3,        --NB
      10        --DUREE
    )

    ADD(MAJ_FUSEES, F)
  END
END
FUSEES=MAJ_FUSEES
END

```

Et ensuite dans les fonctions `_INIT` et `_DRAW`.

```

FUNCTION _INIT()
  FUSEES={}
  FUSEES_MAX=4

```

```

FUSEE_EXPLOSE=FALSE

COULEURS_EXPLOSION={
  {10,9},
  {11,3},
  {8,2},
  {12,1}
}

PARTICULES={}
END

FUNCTION _DRAW()
  IF FUSEE_EXPLOSE THEN
    CLS(13)
  ELSE
    CLS(0)
  END
  CLS()

  DRAW_PARTICULES()
END

```

Tu as vu que maintenant, la fonction `CLS` reçoit un paramètre. C'est optionnel. Quand on n'en passe pas, l'écran sera juste effacé. Mais avec une couleur en paramètre (une valeur entre 0 et 15), l'écran est nettoyé avec cette couleur. D'où le flash gris parce que c'est la couleur correspondant à la valeur 13.

Mais tu sais ce qui pourrait rendre l'explosion encore plus impressionnante, c'est si elle faisait trembler l'écran !

On va avoir besoin d'un nouvel onglet que l'on appellera SECOUSSES. Clique sur le petit + en haut à gauche et ajoute ceci.

```

--SECOUSSES
FUNCTION UPDATE_SECOUSSES()
  IF SEC_FORCE>0 THEN
    LOCAL SEC_X=
      RND(SEC_FORCE)-(SEC_FORCE/2)
    LOCAL SEC_Y=

```

```

    RND(SEC_FORCE)-(SEC_FORCE/2)

    --DECALAGE DE LA CAMERA
    CAMERA(SEC_X, SEC_Y)

    --REDUCTION DE LA SECOUSSE
    -- ET RETOUR A LA NORMAL
    SEC_FORCE=SEC_FORCE*0.9
    IF SEC_FORCE<0.3 THEN
        SEC_FORCE=0
    END
END
END

```

Il faut ensuite apporter des petites modifications dans les fonctions `_INIT` et `_UPDATE`.

```

FUNCTION _INIT()
    FUSEES={}
    FUSEES_MAX=4
    FUSEE_EXPLOSE=FALSE

    COULEURS_EXPLOSION={
        {10,9},
        {11,3},
        {8,2},
        {12,1}
    }

    PARTICULES={}

    SEC_FORCE=0
END

FUNCTION _UPDATE()
    UPDATE_FUSEES()
    UPDATE_PARTICULES()
    UPDATE_SECOUSSES()

    --LANCER UN FEU D'ARTIFICE

```

```

IF BTNP(☒)
  AND FUSEES_MAX>#FUSEES
THEN
  LANCER_FUSEE(
    63,          -- X_DEPART
    127,         -- Y_DEPART
    RND(127-40)+20, -- X_CIBLE
    RND(10)+15,   -- Y_CIBLE
    50,          -- DUREE
    COULEURS_EXPLOSION[
      FLR(RND(4))+1
    ],          -- COULEUR
    TRUE        -- TRAINEE
  )
END
END

```

Et enfin faire une dernière modification dans la fonction `UPDATE_FUSEES`.

```

FUNCTION UPDATE_FUSEES()
  FUSEE_EXPLOSE=FALSE
  LOCAL MAJ_FUSEES={}
  FOR F IN ALL(FUSEES) DO
    --DUREE DE VIE
    F.T=F.T+1
    IF F.T>F.DUREE THEN
      --EXPLOSION DE LA FUSEE
      FUSEE_EXPLOSE=TRUE
      SEC_FORCE=3
      EXPLOSION(
        F.X,      -- X
        F.Y,      -- Y
        F.COULEUR, -- C_TABLE
        20,       -- NB
        TRUE      -- TRAINEE
      )
    ELSE
      --DEPLACEMENT DE LA FUSEE
      F.X=F.X+F.DX
    END
  END
END

```

```

F.Y=F.Y+F.DY

TRAINEE(
    F.X,      --X
    F.Y,      --Y
    1,        --L
    {10,9,8,13}, --C_TABLE
    3,        --NB
    10        --DUREE
)

ADD(MAJ_FUSEES,F)
END
END
FUSEES=MAJ_FUSEES
END

```

Et maintenant, le résultat. Que se passe-t-il maintenant ? Ça tremble quand ça explose !

Tout cela est possible grâce à la fonction `CAMERA`. Elle permet de bouger ... la caméra. On lui passe en paramètre la position x et y où devra se retrouver la partie tout en haut à gauche de la caméra. Par exemple, en appelant `CAMERA(5,30)`, on décale la caméra de 5 pixels sur la droite et de 30 pixels vers le bas (pour rappel, l'axe vertical est à l'envers dans pico8). Et si on appelle `CAMERA(0,0)`, la caméra retourne à son point d'origine.

Dans notre cas, on passe une valeur aléatoire jusqu'à la force de secousse voulu moins la moitié de cette même force. Ça revient à faire bouger la caméra dans l'importe quelle direction, plus ou moins fort selon la force de la secousse. Et à chaque image calculée, on multiplie la force de la secousse par 0,9. En faisant ça, on la diminue un peu à chaque fois jusqu'à ce qu'on atteigne le seuil de 0,3 où là, on arrête la secousse.

## C'est la fin !

Voilà c'est tout pour notre atelier, on peut faire encore plus de choses, les possibilités sont infinies tant que ça ne rame pas trop. Tu peux si tu le souhaites jouer avec les différentes valeurs qu'il y a dans le code et observer ce que ça change ensuite.

Mais avant de faire cela, je te conseille d'enregistrer le projet. J'ai dit tout au début que l'on ne peut pas enregistrer dans la version éducative de Pico8. Mais en fait on peut télécharger le projet et le garder de côté. Pour cela, il faut aller sur l'invité de commande et taper la commande suivante.

## SAVE FEUXARTIFICE

Tu peux donner le nom que tu veux. Ça va télécharger un fichier avec ton projet. D'ailleurs, si tu le souhaites, tu peux l'ouvrir avec n'importe quel éditeur de texte, c'est très lisible.



```
pico-8 cartridge // http://www.pico-8.com
version 42
__lua__
function _init()
  particules={}
  fusees={}
  fusees_max=3
  fusee_explose=false
  couleurs_explosion={
    {10,9},
    {11,3},
    {8,2},
    {12,1}
  }
  sec_force=0
end

function _update60()
  update_fusees()
  update_particules()
  update_secousses()

  if btnp(1)
    and fusees_max>#fusees
  then
    lancer_fusee(
      63,          -- x_depart
      127,         -- y_depart
      rnd(127-40)+20, -- x_cible
      rnd(10)+15    -- v_cible
    )
  end
end
```

Il est possible de charger un projet aussi avec la commande suivante. Mais attention, ça ne fonctionne que depuis le même navigateur. Si tu en utilises un autre ou que tu passes en navigation privé, ça ne fonctionnera pas.

## LOAD FEUXARTIFICE

Un dernier truc cool. Si tu veux voir quelques projet, tu peux installer simplement quelques démos avec cette commande.

## INSTALL\_DEMOS

Ça va télécharger et installer des démos dans un répertoire DEMOS. Tu peux t'y rendre en tapant cette commande

```
CD DEMOS
```

Voir ce que contient ce répertoire avec la commande

```
LS
```

Et enfin en chargeant une des démos en tapant la commande

```
LOAD <nom de la démo>
```

Et ainsi, tu verras que l'éditeur de code affichera le code de la démo et que tu pourras le lancer en faisant CTRL+R.

Amuse toi bien ;)