

Trick or treats - Game Dev

État | Terminé

🎃 Tutoriel

🎃 Création du jeu

🎃 Pour aller plus loin

Viens créer ton jeu d'aventure

Rétrogaming - Fantasy console 8 bits TIC-80 - langage Lua



Lore

> À **Epitech**, l'esprit d'Halloween bat son plein, mais cette année, **Sombrenuit**, un farceur légendaire, a subtilisé tous les bonbons ! Pour les récupérer, les courageux participants du **"Halloween Camp"** devront coder leurs propres

> aventures, débugger des lignes de code pour explorer tous les recoins hantées du monde qu'ils auront imaginé. Équipé de ton sac à bonbons, ton héro combattra des monstres, explorera des donjons, et affrontera **Sombrenuit** pour restituer les friandises à **Epitech** et à ses valeureux étudiants.

Matériel nécessaire & installation

Les participants viennent avec un ordinateur (PC, MAC, Linux), il n'y a pas de configuration minimum, le logiciel TIC-80 ne nécessite pas un ordinateur puissant ou récent (On peut prêter des ordinateurs, attention à bien gérer les demandes en amont pour que nous puissions anticiper)

TIC-80 peut se lancer directement dans le navigateur, ce qui veut dire qu'il n'y a pas de temps à prévoir pour l'installation. On peut aussi télécharger une version à installer sur l'ordinateur du participant.

TIC-80 est un logiciel libre (Licence MIT), les participants peuvent donc l'installer et l'utiliser gratuitement.

Avantages de TIC-80

- C'est super fun car on peut tout de suite jouer et customiser son jeu.
- En une heure, un débutant pour prendre en main l'environnement, cela va lui permettre de coder, créer les sprites, les maps, les effets sonores et la bande son.
- On code pour de vrai sans se prendre trop la tête : boucles, conditions, variables

- On peut publier sur un site les jeux crées en quelques clics, ce qui permet de faire une démonstration à ses proches.



Tutoriel

Lore

> À **Epitech**, l'esprit d'Halloween bat son plein, mais cette année, **Sombrenuit**, un farceur légendaire, a subtilisé tous les bonbons !
> Pour les récupérer, les courageux participants du **"Halloween Camp"** devront coder leurs propres aventures, débugger des lignes de code pour explorer tous les recoins hantées du monde qu'ils auront imaginé.
> Équipé de ton sac à bonbons, ton héro combattra des monstres, explorera des donjons, et affrontera **Sombrenuit** pour restituer les friandises à **Epitech** et à ses valeureux étudiants.

La méthode Epitech : Learn different

- Vous allez acquérir des nouvelles compétences en pratiquant et en vous amusant.
- Votre premier réflexe lorsque vous vous posez une question doit être de demander de l'aide à vos voisins. Si vous ne trouvez pas la solution demandez de l'aide à un accompagnateur.
- Pour avancer sereinement dans la réalisation de votre projet, pensez à faire valider chaque partie au fur et à mesure par un animateur Epitech qui pourra vous donner des conseils

- Nous vous encourageons à utiliser les resources disponibles sur internet pour trouver des réponses à vos questions et pour vous renseigner.
- Vous pouvez ajouter des fonctionnalités de votre choix et personnaliser votre projet : c'est ce que nous appelons un bonus à Epitech et c'est possible pour tous les projets
- Les IA génératives peuvent sûrement faire cet atelier, mais si vous les utilisez vous n'apprendrez rien.



Pré-requis

TIC-80 est une fantasy retro console open source conçu pour créer, jouer et partager de petits jeux. Prenez quelques instants pour la découvrir en testant un jeu : https://tic80.com/play

Tous les outils sont intégrés pour vous permettre de créer facilement un jeu : éditeurs de code, de sprites, de cartes, de sons, ainsi qu'un terminal.

Pour réaliser cette atelier, vous pouvez utiliser directement TIC-80 dans votre navigateur : https://tic80.com/create ou télécharger une version à installer sur votre ordinateur depuis https://tic80.com/

Reset "hello world"

Le plus simple pour bien comprendre le fonctionnent de TIC-80 consiste de partir d'un environnement vierge, nous allons donc supprimer tous les éléments de la démo.

- Utilisez la touche `ESC` de votre clavier pour vous rendre dans l'éditeur de code
- Vous pouvez sélectionner tout texte avec `CTRL` + `A` et ensuite supprimer le code existant
- Utilisez la touche `F2` (`fn` + `F2` sur Mac) de votre clavier pour vous rendre dans l'éditeur de **sprite**

- Vous pouvez ensuite utiliser le **canvas zoom** pour sélectionner la zone avec tous les **sprites** et supprimer les **sprites** existants (touche 8)

Pour vous aider, vous pouvez vous référer à la vidéo ci-dessous !

attachment:8ded5255-431b-4c93-908b-5df63cfada20:Capture_vido_du_2024-10-09_22-26-46.webm

Créer mon premier mini jeu

Vous êtes prêt à créer votre premier mini jeu

Création d'un nouveau sprite

Rendez vous dans l'éditeur de **sprite,** dé-zoomez au maximum et dessinez votre premier **sprite**, prenez le temps d'essayer les différents outils à votre disposition, cela vous sera utile pour la suite.



📌 Vous remarquerez que chaque **sprite** ne fait que 8×8 et que la palette de couleur est limitée à 16 couleurs



Affichage du sprite

Nous allons maintenant retourner dans l'éditeur de code et écrire le code suivant :

```
-- il est possible de mettre des commentaires
-- pour expliquer une partie du code
```

```
-- un commentaire commence toujours par deux tirets --

function TIC()
 cls() -- efface l'ecran de jeu
 spr(1, 80, 80, 0) -- affiche le sprite #1 a la position 80, 80
end
```

Vous pouvez maintenant retourner sur le terminal avec la touche `ESC` de votre clavier et taper la commande `run` pour lancer votre jeu

> 📌 Le dernier argument de `spr` permet d'indiquer à TIC-80 que l'on souhaite utiliser la couleur 0 (noir) comme couleur de **transparence**.

Si vous avez réalisé cette étape correctement, vous devriez obtenir un résultat similaire à ceci lorsque vous lancerez votre jeu :



Faire bouger à l'écran mon sprite

Pour l'instant la position de notre **sprite** sera toujours 80 pixels à droite et 80 pixels vers le bas **à partir du coin supérieur gauche**

Pour changer ce mode de fonctionnement, nous allons devoir créer deux nouvelles variables `player_x` et `player_y` qui vont vous permettre de changer dynamiquement la position du joueur.

```
player_x = 80
player_y = 80
```

```
-- la fonction TIC() se trouve en dessous de ce code
```

Nous allons maintenant utiliser la fonction `btn` et ajouter une condition grâce à une structure de contrôle `if` pour modifier les coordonnées du **sprite** et pour le faire bouger à l'écran

```
function TIC()
 -- si le bouton 0 (touche haut du clavier) est appuyée,
 -- on veut que le joueur monte ⬆️
 -- Il faut alors soustraire 1 pixel
 -- à la position du joueur sur l'axe des ordonnées : y
 if btn(0) then player_y = player_y - 1 end

 cls() -- efface l'ecran de jeu
 spr(1, player_x, player_y, 0) -- affiche le sprite #1 relatif aux coordonnées du j
end
```

Exercice 1 : Faites bouger votre sprite de joueur dans toutes les directions

En vous inspirant du code précédent, vous devez faire en sorte que votre sprite de joueur puisse bouger dans les 4 directions ⬆️⬇️⬅️➡️.

> 📌 Consultez le wiki officiel en ligne pour vous aider : https://github.com/nesbox/TIC-80/wiki

▼ Indice

La fonction `btn` prend en paramètre un nombre (`0` : flèche du haut ⬆️ du clavier). Une page dans l'aide contient le tableau de correspondance avec les touches du clavier : https://github.com/nesbox/TIC-80/wiki/key-map

▼ Solution

```
player_x = 60
player_y = 80

function TIC()
 -- le bouton 0 correpond a la fleche du haut du clavier
 if btn(0)) then player_y=player_y-1 end
 if btn(1)) then player_y=player_y+1 end
 if btn(2)) then player_x=player_x-1 end
 if btn(3)) then player_x=player_x+1 end

 cls() -- efface l'ecran de jeu
 spr(1, player_x, player_y, 0) -- affiche le sprite #1 a la position 60,80
end
```

Exercice 2 : Fixez des limites

Vous avez sûrement remarqué que votre **sprite** de joueur peut sortir de l'écran. Trouvez un moyen de limiter ses déplacements à l'intérieur de l'écran visible... pour ne pas qu'il s'échappe de votre ordinateur ! 🏃

📌 La partie visible de l'écran fait 248 pixels par 144 pixels... mais attention à la taille de votre sprite !

▼ Indice

Utilisez une structure de contrôle `if` et les conditions inférieur à `<` et supérieur à `>`

```
-- si la coordonnée x (horizontale) du joueur est en dessous de 0,
-- alors on fixe la coordonnée x à zero
if player_x < 0 then player_x = 0 end
```

▼ Solution

```
function TIC()
 -- au dessus de ce commentaire tout ce qui concerne les deplacements d
```

```
 if player_x + 8 > 240 then player_x = 240 - 8 end
 if player_x < 0 then player_x = 0 end
 if player_y + 8 > 136 then player_y = 136 - 8 end
 if player_y < 0 then player_y = 0 end

 -- en dessous de ce commentaire tout ce qui concerne l'affichage du spr
end
```

Exercice 3 : Affichage de la carte

En ajoutant une carte à votre mini jeu, vous ajoutez une ambiance dans votre jeu, la carte va s'afficher en arrière plan et dans une prochaine étape nous apprendrons à interagir avec les éléments de la cartes : collision avec les objets et les bordures, récupération d'items

la premier étape consiste à ajouter la ligne de code pour afficher la carte

```
function TIC()
-- au desssus de ce commentaire tout ce qui concerne les calculs
cls()
map() -- affichage de la carte après avoir fait le clear screen
-- en dessous de ce commentaire tout ce qui concerne l'affichage
end
```

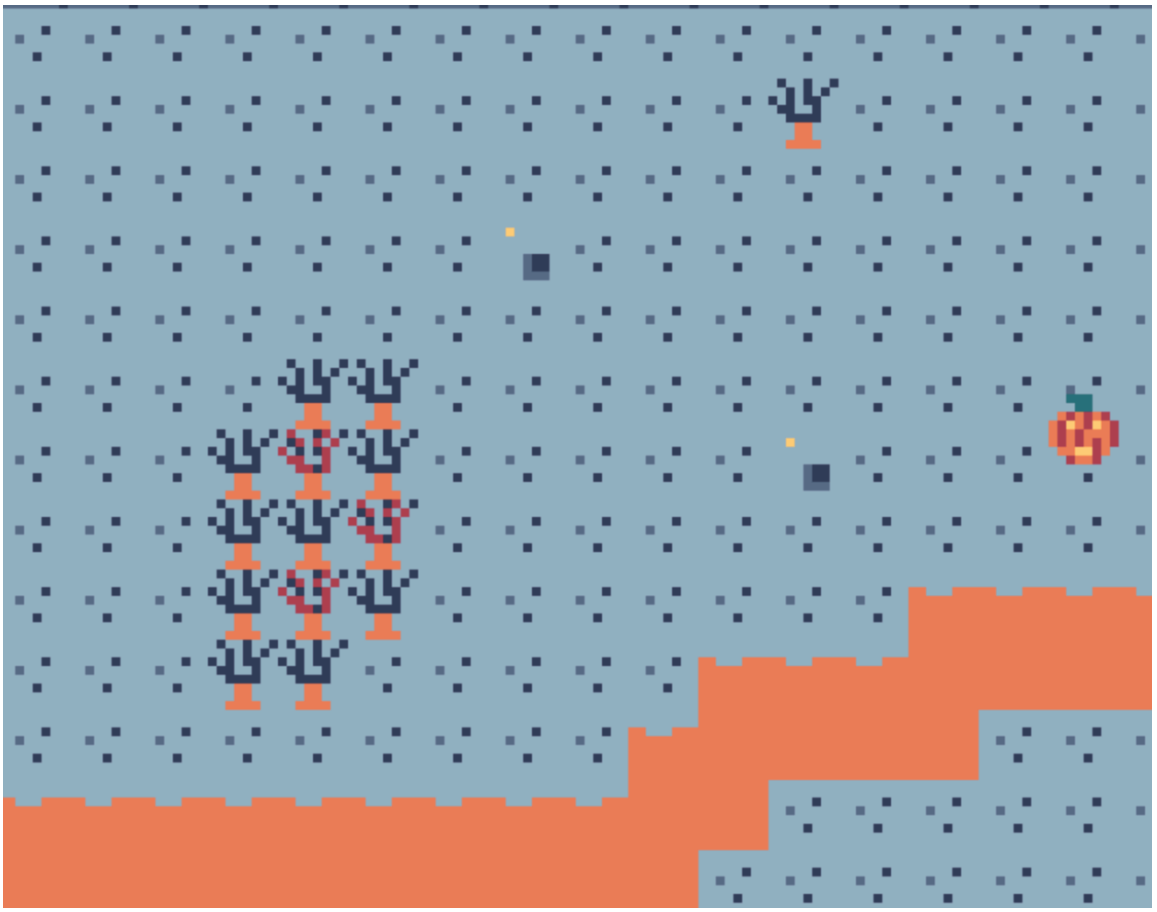
Etant donné que vous commencez à être plus à l'aise avec TIC-80, l'objectif de cet exercice va être de passer de l'éditeur de sprite à l'éditeur de carte pour ajouter un fond et un décor.

- Créez des sprites avec des motifs différents (texture) que vous pourrez utiliser pour le fond de votre carte
- Créez des sprites avec des objets pour faire les décors : arbres, rochers, une rivière
- Ajoutez ces sprites à votre carte et testez votre jeu

📌 Libérez vos talents d'artiste, dessinez en 8 bits est à la portée de toutes et tous !

▼ Solution





Exercice 4 (facultatif): Organisez votre code

Pour rendre plus lisible et réutilisable votre code, vous pouvez créer des objets et vos propres fonctions. A partir de votre code existant :

- créez une fonction `move_player` qui va s'occuper des déplacements du joueur
- créez une fonction `map_limit` qui va s'occuper de bloquer le joueur à l'intérieur de la carte
- créez une fonction `draw` qui va s'occuper de l'affichage
- créez un objet `p` (pour « player ») qui va contenir toutes les informations sur le joueur

📌 Il est intéressant d'échanger avec vos voisins et avec les encadrants sur les notions d'**organisation de code**, de **fonction** et d'**objet**.

▼ Exemple de solution

```
p = {x = 80, y = 80, sprite = 1}

function move_player()
 if btn(0) then p.y = p.y - 1 end
 if btn(1) then p.y = p.y + 1 end
 if btn(2) then p.x = p.x - 1 end
 if btn(3) then p.x = p.x + 1 end
end

function map_limit()
 if p.x + 8 > 240 then p.x = 240 - 8 end
 if p.x < 0 then p.x = 0 end
 if p.y + 8 > 136 then p.y = 136 - 8 end
 if p.y < 0 then p.y = 0 end
end

function draw()
 cls()
 map()
 spr(p.sprite,p.x,p.y,0)
end

function TIC()
 move_player()
 map_limit()
 draw()
end
```

Crédits

Cet atelier a été écrit et testé à Epitech Montpellier 💙 pour le Coding Club

Merci à Game Dev Alliance, une communauté francophone de développeuses et développeurs indépendants, et particulièrement à Aurelien pour l’inspiration grâce à ses tutos et ses vidéos de création de jeux sur PICO-8 (la grande soeur de TIC-80)



Création du jeu

Pré-requis

Avant de vous lancer dans cet atelier, réalisez Tutoriel qui sera notre base pour la création de votre jeu d'aventure **Trick or Treats.**

Pré-requis
Créer ton jeu d'aventure
- Se déplacer
- Gérer les collisions
  - Étape 1 : Définissez des nouvelles variables
  - Étape 2 : Assignez un flag
  - Étape 3 : (difficile) Créez la fonction `check_flag`
  - Étape 4 : Vérifier si le joueur peut se déplacer
- Déplacer la caméra
  - Version 1 - écran par écran
  - Version 1 - Étape 1 : Dessiner un nouveau monde
  - Version 1 - Étape 2 : Déterminer la position de la caméra
  - Version 1 - Étape 3 : Fonctions d'affichage
  - Version 2 - suivez moi
  - Version 2 - Étape 1 : Dessiner un nouveau monde
  - Version 2 - Étape 2 : Déterminer la position de la caméra
  - Version 2 - Étape 3 : Fonctions d'affichage

Crédits

Créer ton jeu d'aventure

Se déplacer

Il faut considérer la carte comme un plateau de jeu avec des tuiles (tile) de 8 pixels par 8 pixels

Pour simplifier la création de jeu, nous allons faire en sorte que le joueur se déplace de tuile en tuile. Vous allez devoir modifier la fonction `move_player` pour déplacer le sprite du joueur de 8 pixels en 8 pixels, pour obtenir l'effet voulu, préférez l'utilisation de la fonction `btnp` à la place de `btn`.

```
p = {x = 8, y = 8, sprite = 1}

-- dans ma fonction move_player()
if btnp(0) then p.y = p.y - 8 end
if btnp(1) then p.y = p.y + 8 end
if btnp(2) then p.x = p.x - 8 end
if btnp(3) then p.x = p.x + 8 end
```

Gérer les collisions

Pour gérer le collisions, vous allez utiliser des fonctionnalités très pratiques disponibles dans TIC-80.

Vous allez devoir passer par les étapes suivantes :

1. Définir des nouvelles variables dans la fonction `move_player` pour pouvoir vérifier ce qui se trouve dans la tuile suivante avant de déplacer le joueur
2. Assigner un flag aux sprites avec lesquels on souhaite que le joueur rentre en collision
3. Créer une fonction `check_flag` qui va renvoyer `true` si le flag est assigné à un sprite
4. Utiliser la fonction `check_flag` dans la fonction `move_player` pour décider si le joueur se déplace ou non en fonction de ce qui se trouve dans la prochaine tuile

Étape 1 : Définissez des nouvelles variables

Dans la fonction `move_player`, ajoutez deux nouvelles variables `new_x` et `new_y` qui vont permettre de réaliser des vérifications avant déplacer le joueur

▼ Solution

```
new_x = p.x
new_y = p.y
```

```
if btnp(0) then new_y = new_y - 8 end
if btnp(1) then new_y = new_y + 8 end
if btnp(2) then new_x = new_x - 8 end
if btnp(3) then new_x = new_x + 8 end

p.x = new_x
p.y = new_y
```

Étape 2 : Assignez un flag

Assignez le flag 0 à **tous les sprites** avec lesquels vous souhaitez que le joueur entre en collision

▼ Indice

Utilisez les fonctions avancées en activant l'interrupteur en haut à gauche de l'éditeur, et cliquez sur le flag que vous souhaitez assigner à votre sprite

▼ Solution

attachment:d7a3f08e-8429-4896-8108-bdcf93ee46ea:Capture_vido_du_2024-10-10_09-27-03.webm

Étape 3 : (difficile) Créez la fonction `check_flag`

La fonction `check_flag` prend en argument les coordonnées x et y du pixel en haut à gauche du sprite sur la carte et le numéro du flag à vérifier et renvoie `true` si le flag est assigné au sprite en question et `false` si ce n'est pas le cas.

📌 Cette fonction est très pratique et pourra être réutiliser plus tard dans votre code

▼ Indice

Pour vous aider

- La fonction `mget(x,y)` renvoie le sprite_id qui se trouve en x et y de la carte

- La fonction `fget(sprite_id,flag)` qui renvoie `true` si le sprite en question possède ce flag
- le mot clef `return` permet à une fonction de retourner une valeur

ci-dessous un patron de la fonction à implémenter :

```
function check_flag(x, y)
 -- enlevez ce commentaire et implementez la fonction
end
```

▼ Solution

```
function check_flag(x,y,flag)
 return fget(mget(x,y), flag)
end
```

Étape 4 : Vérifier si le joueur peut se déplacer

Vous avez tous les éléments pour pouvoir implémenter les collisions dans votre fonction `move_player`

📌 Attention à ne pas confondre les tuiles et les pixels !

▼ Indice

`mget(x, y)` attend en paramètre une coordonnée d'une tuile de la carte mais les coordonnées du joueur sont en pixels !

Il faudrait trouver un moyen de **calculer** les coordonnées de la tuile à partir des coordonnées du joueur.

▼ Solution

```
function move_player()
 new_x = p.x
 new_y = p.y

 if btnp(0) then new_y = new_y - 8 end
 if btnp(1) then new_y = new_y + 8 end
```

```
 if btnp(2) then new_x = new_x - 8 end
 if btnp(3) then new_x = new_x + 8 end

 if not check_flag(new_x / 8, new_y / 8,0) then
  p.x = new_x
  p.y = new_y
 end
end
```

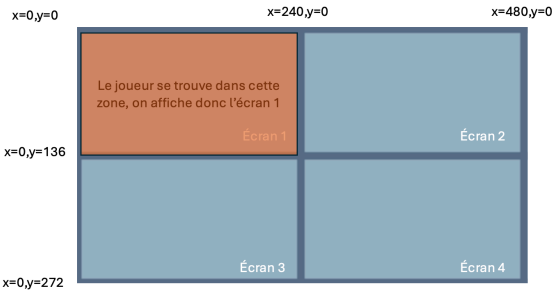
Déplacer la caméra

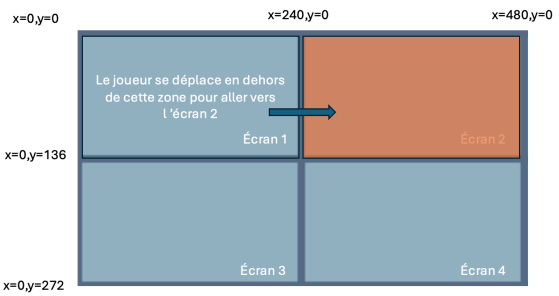
Maintenant que vous avez mis en place les mécanismes de base de votre jeu, pour rendre le monde plus vaste à explorer. Il existe plusieurs manières de déplacer la caméra, voici deux versions de la caméra, choisissez celle qui vous semble le plus adaptées :

1. Une caméra qui va se déplacer d'écran en écran (exemple Binding of Isaac, Zelda sur Nes)
2. (difficile) Une caméra ce qui va vous permettre de suivre le joueur qui veut se déplacer en dehors du premier écran.

Version 1 - écran par écran

Dans cet exemple, le monde est composé de 4 écrans, si le joueur se trouve dans la zone orange alors, on affiche cette partie de la carte





Pour implémenter ce monde et la caméra, vous allez devoir passer par les étapes suivantes :

1. Dessiner un monde sur plusieurs écrans dans l'éditeur de carte

2. Créer une fonction qui va déterminer la position de la caméra en fonction de celle du joueur
3. Modifier les fonctions d'affichage pour prendre en compte la caméra

Version 1 - Étape 1 : Dessiner un nouveau monde

Rendez vous dans l'éditeur de carte, utilisez les fonctions de zoom et les différents outils pour créer votre monde, vous pouvez aussi créer des nouveaux sprites

> 📌 N'oubliez pas de mettre les bons flags pour gérer les collisions !

Version 1 - Étape 2 : Déterminer la position de la caméra

Vous allez devoir créer :

- un nouvelle variable camera qui sera un objet qui contient la position de la camera
- une fonction qui va déterminer la position de la camera en fonction de la position du joueur, de sorte à ce qu'il puisse se déplacer sur les différents écrans

> 📌 Attention, la fonction `map_limit` vous empêche actuellement de dépasser l'écran

▼ Solution

```
cam = {x = 0, y = 0}

function camera()
 -- on determine d'abord sur quel "ecran" le joueur est par rapport
 -- a ses coordonnees
 local screen_x = math.floor(p.x / 239)
 local screen_y = math.floor(p.y / 135)
```

```
 -- on calcule le point du rectangle qui represente la camera sur la map
 cam.x = screen_x * 30
 cam.y = screen_y * 17
end
```

Version 1 - Étape 3 : Fonctions d'affichage

Vous allez modifier la fonction d'affichage de la carte et la fonction d'affichage du sprite du joueur

> 📌 Le joueur doit être dans le même référentiel que celui de la caméra.

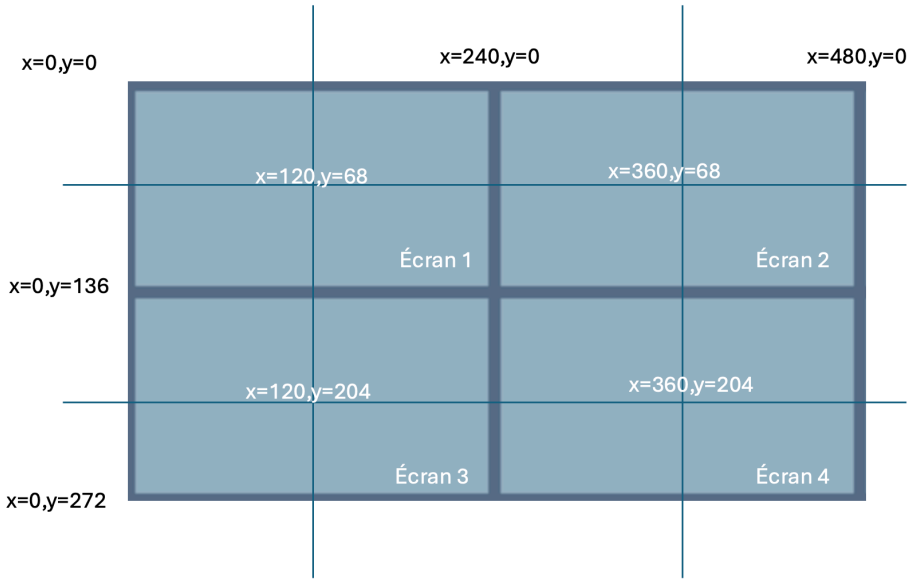
Pour vous aider, https://github.com/nesbox/TIC-80/wiki/map

▼ Solution

```
function draw()
 map(cam.x, cam.y)
 spr(p.sprite, p.x % 240, p.y % 136, 0)
end
```

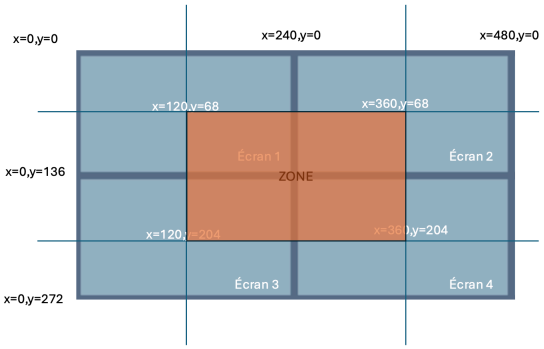
Version 2 - suivez moi

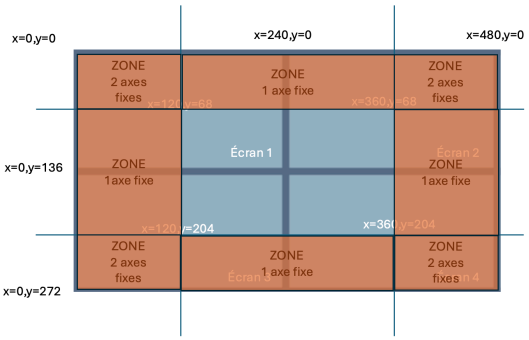
Dans cet exemple, le monde est composé de 4 écrans



Le joueur se déplace dans la zone du milieu du monde, alors la caméra se déplace en suivant le joueur

Le joueur se déplace dans la zone qui borde le monde, alors la caméra devient fixe sur 1 ou 2 axes





Pour implémenter ce monde et la caméra, vous allez devoir passer par les étapes suivantes :

1. Dessiner un monde sur plusieurs écrans dans l'éditeur de carte
2. Créer une fonction qui va déterminer la position de la caméra en fonction de celle du joueur
3. Modifier les fonctions d'affichage pour prendre en compte la caméra

Version 2 - Étape 1 : Dessiner un nouveau monde

Rendez vous dans l'éditeur de carte, utilisez les fonctions de zoom et les différents outils pour créer votre monde, vous pouvez aussi créer des nouveaux sprites

> 📌 N'oubliez pas de mettre les bons flags pour gérer les collisions !

Version 2 - Étape 2 : Déterminer la position de la caméra

Vous allez devoir créer :

- un nouvelle variable camera qui sera un objet qui contient la position de la camera
- une fonction qui va déterminer la position de la camera en fonction de la position du joueur

▼ Solution

```
cam={x=0,y=0}

function camera()
if (p.x>120 and p.x<360) then
   cam.x=p.x-120
elseif (p.x<120) then
   cam.x=0
elseif (p.x>=360) then
   cam.x=240
end

if (p.y>68 and p.y<204) then
   cam.y=p.y-68
elseif (p.y<68) then
   cam.y=0
elseif (p.y>=204) then
   cam.y=136
end
end
```

Version 2 - Étape 3 : Fonctions d'affichage

Vous allez modifier la fonction d'affichage de la carte et la fonction d'affichage du sprite du joueur

> 📌 Le joueur doit être dans le même référentiel que celui de la caméra.

Pour vous aider, https://github.com/nesbox/TIC-80/wiki/map

▼ Solution

```
function draw()
map(0,0,240,136,-cam.x,-cam.y)
spr(p.sprite,p.x-cam.x,p.y-cam.y,0)
end
```

Crédits

Cet atelier a été écrit et testé à Epitech Montpellier 💙 pour le Coding Club

Merci à Game Dev Alliance, une communauté francophone de développeuses et développeurs indépendants, et particulièrement à Aurelien pour l'inspiration grâce à ses tutos et ses vidéos de création de jeux sur PICO-8 (la grande soeur de TIC-80)



Pour aller plus loin

Pré-requis

Avant de commencer cette partie, Réaliser la 🎃 Création du jeu pour créer votre jeu d'aventure. Ce nouvel atelier a pour objectif d'ajouter nouvelles fonctionnalités à votre jeu : collecte d'objets, boite de dialogue. Vous allez retrouver des mentions **A vous de coder !** qui vont vous indiquer que vous devez vous lancer dans l'implémentation de la nouvelle fonctionnalité. Nous vous recommandons vivement de lire toutes les étapes liées à une fonctionnalité et de comprendre le mode de fonctionnement avant de vous lancer.

Pré-requis
- Ramasser des bonbons
  - Etape 1 : Sprite bonbon miam miam
  - Etape 2 : Ramasser et compter
  - Etape 3 : Interagir
  - Etape 4 : Afficher le score "bonbon" dans l'interface
- Créer des boites de dialogue
  - Etape 1 : Affichage des boites de dialogue
  - Etape 2 : Création et mise à jour d'un message
  - Etape 3 : Dialogues dans le jeu
- Finisher
- Pour aller plus loin

Crédits

Ramasser des bonbons

Pour collecter des bonbons (et de manière générale des items) vous allez devoir passer par les étapes suivantes

1. Dessiner les sprites items "bonbon" et créer une fonction `next_tile` pour pouvoir modifier un sprite pendant le jeu (pour faire disparaitre les bonbons

une fois ramassé)

2. Créer une fonction `pickup_candy` qui va s'occuper de toute la mécanique de ramassage des bonbons
3. Créer une fonction `interact` pour gérer les interactions avec les items (dans notre cas les bonbons)
4. Et enfin, créer une fonction `draw_ui` qui va afficher un compteur de bonbons ramassés

Etape 1 : Sprite bonbon miam miam

Dans l'éditeur de sprite, créez deux sprites avec des `#ID` qui se suivent pour faciliter le mécanisme de modification de sprite après une interaction.

Le premier sprite représente le bonbon (par exemple avec l'ID `#4`) et le deuxième sprite représente une tuile vide après ramassage du bonbon (par exemple avec l'ID `#5`)

Vous allez ensuite créer la fonction `next_tile` qui va remplacer le sprite actuel sur la carte par le suivant dans la palette des sprites.

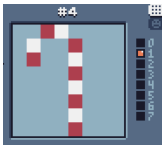
> 📌 Ce mécanisme est une astuce pour gérer facilement les sprites de type item, vous pourrez réutiliser cette fonction pour mettre en place rapidement facilement des pièces d'or, des coffres, des portes...

A vous de coder !

▼ Indice

Utilisez les fonction `mset` et `mget`

▼ Solution

Sprite N	**Sprite N+1**
	

```
function next_tile(x,y)
   mset(x,y,mget(x,y)+1)
end
```

Etape 2 : Ramasser et compter

Créez une fonction `pick_candy` qui a pour rôle de gérer les spécificités du ramassage et du comptage de bonbon dans votre jeu. Dans notre cas, le bonbon disparait et le compteur de bonbon augmente d'une unité.

> 📌 On pourrait imaginer d'autres mécanismes comme retirer une unité au compteur de bonbon si le joueur ramasse un bonbon empoisonné

A vous de coder !

▼ Indice

Utilisez la fonction `next_tile(x, y)` pour remplacer le sprite du bonbon par un sprite de case vide après ramassage, et n'oubliez de créer la variable compteur de bonbons `p.candy`.

▼ Solution

```
p={x=64,y=64,sprite=1,candy=0}

function pickup_candy(x,y)
next_tile(x,y)
p.candy=p.candy+1
end
```

Etape 3 : Interagir

Créez la fonction `interact` qui va vérifier si un sprite possède le flag correspondant aux bonbons et qui va appeler la fonction `pickup_candy` que vous avez implémenté dans l'étape précédente.

Il vous reste juste à appeler la fonction `interact` dans votre fonction `move_player`

📌 En appelant la fonction `interact` avant les collisions et en utilisant les flag correctement, on va pouvoir plus tard implémenter facilement des mécaniques comme l'ouverture d'une porte 🤯

A vous de coder !

▼ Indice

Utilisez la fonction `check_flag(x, y, 1)` pour déterminer si le sprite avec lequel le joueur interagit a le bon flag (celui assigné aux bonbons).

▼ Solution

```
function interact(x,y)
 if check_flag(x,y,1) then pickup_candy(x,y) end
end

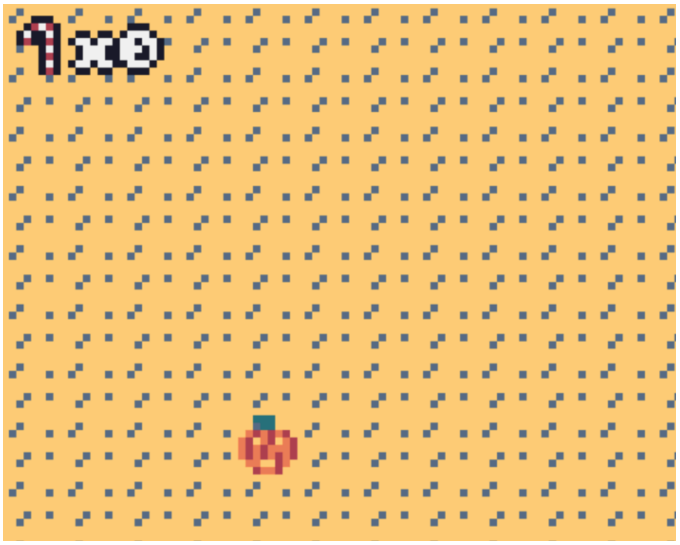
function move_player()
-- le code qui gère les flèches du clavier

interact(new_x/8,new_y/8)

-- le code qui gère les collisions
end
```

Etape 4 : Afficher le score "bonbon" dans l'interface

Créez un fonction `draw_ui` (pour User Interface) qui va permettre d'afficher le score en haut à gauche de votre écran et appelez cette fonction dans votre code après la fonction `draw` car on veut que le score soit toujours au dessus de tous les autres éléments. L'objectif est d'obtenir le résultat qui pourrait ressembler à cette capture de jeu :



A vous de coder !

▼ Indice

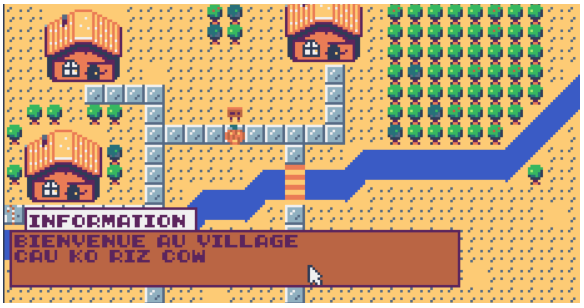
- Créez un sprite identique au sprite du bonbon mais avec une autre couleur de transparence pour pouvoir utiliser le noir comme contour
- Pour faire un effet de contour noir autour du texte, affichez 4 fois le texte avant le texte souhaité avec respectivement un pixel de décalage à droite, à gauche, en haut, en bas

▼ Solution

```
function draw_ui()
spr(52,2,2,11)
-- Tips : pour faire un effet de contour noir autour du texte
-- on affiche 4 fois le texte avant le texte souhaité
-- avec un pixel de décalage à droite, à gauche, en haut, en bas
print("x"..p.candy,10+1,4,0)
print("x"..p.candy,10-1,4,0)
print("x"..p.candy,10,4+1,0)
print("x"..p.candy,10,4-1,0)
print("x"..p.candy,10,4,12)
end
```

Créer des boites de dialogue

Les boites de dialogues vont rendre votre jeu plus interactif et sont très utiles pour interagir avec des objets de la carte, par exemple un panneau



Pour mettre en place des boites de dialogues, vous allez passer par les étapes suivantes :

1. Créer un système d'affichage pour vos boites de dialogues
2. Implémenter un système réutilisable de création et de mise à jour d'un message
3. Ajouter des dialogues dans votre jeu

Etape 1 : Affichage des boites de dialogue

Vous pouvez afficher un texte à l'écran grâce à la fonction `print` . En utilisant les paramètres de cette fonction vous allez positionner ce texte à l'écran et changer sa couleur.

Pour obtenir un rendu qui ressemble à ce qui existe dans un jeu d'aventure, créez un boite de dialogue avec une zone de titre et une zone pour le message.

Enfin, pour rendre réutilisable dans votre code l'affichage des boites de dialogue, créez une nouvelle fonction `draw_msg` et utilisez un tableau `message={}` qui va contenir le titre du message et le message à afficher, par exemple `message={"CITROUILLE","SALUT A TOI"}`

A vous de coder !

▼ Indice

📌 Lorsque vous programmez la majorité des documentations sont en anglais, pour commencer à vous habituer, les indices ci-dessous seront en anglais.

- `rect(x y w h color)` This function draws a filled rectangle of the desired size and color at the specified position.If you only need to draw the the border or outline of a rectangle (ie not filled) see `rectb()`
- `rectb(x y w h color)` This function draws a one pixel thick rectangle border at the position requested. If you need to fill the rectangle with a color, see `rect()` instead.
- `print(text x=0 y=0 color=15)` This will simply print text to the screen using the font defined in config.
- `#` operator in Lua returns the length of a string or table, counting the number of characters or elements, respectively.

▼ Solution

```
message={"CITROUILLE","SALUT A TOI"}

function draw_msg()
    local y=100
    rect(16,y-3,#message[1]*6+3,10,12)
    rectb(16,y-3,#message[1]*6+3,10,1)
    print(message[1],18,y,1)
    rect(10,y+6,180,23,2)
    rectb(10,y+6,180,23,1)
    print(message[2],12,y-2+10,1)
end

function TIC()
-- après l'affichage de la carte et du joueur
draw_msg()
end
```

📌 Une variable locale n'existe que dans la fonction dans laquelle elle est déclarée, échangez avec un accompagnateur si vous souhaitez en savoir plus sur cette notion

Etape 2 : Création et mise à jour d'un message

Pour rendre votre système de dialogue plus simple à utiliser et plus flexible, créez une fonction `create_msg` qui va être appelée pour remplir le tableau `message={}` .

En ajoutant un élément en première position , qui va prendre la valeur `true` ou `false` à votre tableau `message={}` , vous allez contrôler l'affichage du message en cours ce qui va vous permettre de gérer le cas ou un message doit disparaitre après une action, par exemple lors de l'appuie sur la touche `W` du clavier. Vous devriez obtenir le comportement suivant : `create_msg("JACK","SALUT MA CITROUILLE")` qui met à jour le contenu du tableau `messsage={true,"JACK,"SALUT MA CITROUILLE}`

A vous de coder !

▼ Solution

```
function create_msg(name,msg)
   message={true,name,msg}
end
```

Il ne vous reste plus qu'a créer la fonction `update_msg` qui va modifier le premier élément du tableau `message` qui va prendre la valeur `false` pour indiquer que le message doit disparaitre lorsque la touche `W` du clavier est appuyé.

A vous de coder !

▼ Indice

Pensez à appeler votre fonction `update_msg` au bon endroit dans la fonction `TIC`

▼ Solution

```
function update_msg()
   if btn(4) then message[1]=false end
end
```

```
function TIC()
-- avant les fonctions liées à l'affichage
update_msg()
```

Enfin modifiez votre fonction `draw_msg` pour indiquer si le message doit être affiché en fonction de la valeur du premier élément du tableau.

A vous de coder !

▼ Solution

```
function draw_msg()
   local y=100
   -- si le premier element du tableau message est égale à true
   if message[1] then
      rect(16,y-3,#message[2]*6+3,10,12)
      rectb(16,y-3,#message[2]*6+3,10,1)
      print(message[2],18,y,1)
      rect(10,y+6,180,23,2)
      rectb(10,y+6,180,23,1)
      print(message[3],12,y-2+10,1)
   end
end
```

Il est grand temps de tester cette nouvelle fonctionnalité en appelant avec un message personnalisé votre fonction `create_msg` **.** Vous devriez voir apparaitre votre message dans la boite de dialogue et qui va disparaitre lorsque vous appuyez sur la touche `W` du clavier

Etape 3 : Dialogues dans le jeu

Pour intégrer votre système de dialogues dans votre jeu

Dans cette étape vous allez :

- Créer un sprite pour panneau d'affichage
- Afficher une boite de dialogue lorsque vous rentrez en collision votre panneau
- Bloquer les déplacements tant que le joueur n'a pas fait disparaitre la boite de dialogue

A vous de coder !

▼ Indice

1. Utilisez la fonction `interact` que vous avez codé pour votre système de ramassage de bonbons.
2. Les coordonnées du sprite de panneau sur la carte vous seront utiles
3. Le premier élément du tableau `message` permet de contrôler l'affichage de la boite de dialogue, cela peut être utile pour bloquer les déplacements du joueur

▼ solution

```
function interact(x,y)
 -- bonbons
 if check_flag(x,y,1)  then pickup_candy(x,y) end
 -- panneau a la position (12,7) sur la carte
 if (x == 12 and y ==7 ) then create_msg("INFORMATION","BIENVENUE AU
end

function TIC()
-- pour bloquer les déplacement tant que le joueur
-- n'a pas fait disparaitre la boite de dialogue
if not message[1] then move() end
-- en dessous la suite du code de la fonction TIC
end
```

Finisher

▼ Félicitations ! Vous êtes allez plus loin que 99% des participants



Pour aller plus loin

Grâce à tout ce que vous avez appris, vous pouvez ajouter pleins de nouvelles fonctionnalités à votre jeu, par exemple

- Des clefs qui permettent d'ouvrir des coffres
- Un PNJ (personnage non joueur) avec qui vous pouvez parler
- Une fin pour votre jeu

et tout ce qui vous passe par la tête ... amusez vous bien 😄

Crédits

Cet atelier a été écrit et testé à Epitech Montpellier 💙 pour le Coding Club

Merci à Game Dev Alliance, une communauté francophone de développeuses et développeurs indépendants, et particulièrement à Aurelien pour l'inspiration grâce à ses tutos et ses vidéos de création de jeux sur PICO-8 (la grande soeur de TIC-80)