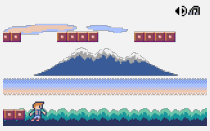
Jump Game



L'objectif de ce second tutoriel est d'implémenter un jeu de saut avec le module Pyxel.
Vous apprendrez quelques techniques indispensables à la création d'un jeu vidéo :

- animer un personnage
- faire sauter un personnage de manière réaliste
- faire défiler un décor pour obtenir un effet de profondeur
- gérer les sons du jeu

Évidemment pour réaliser ce tuto, nous utiliserons les techniques apprises dans le premier tutoriel. N'hésitez pas à relire la feuille de route.

Pour chaque étape de se tutoriel de découverte, retrouvez sur studserver ce que vous devez obtenir.

Vous êtes bien sur libres de prendre des libertés avec ce tutoriel.
N'hésitez pas à tester, à modifier, à améliorer le jeu que vous allez créer en suivant les étapes de ce tutoriel.
Faîtes preuve d'imagination et de créativité !

Comme dans le premier tutoriel vous pouvez :

- directement sur votre machine ;
- en utilisant l'app https://www.pyxelstudio.net/ ;
- en utilisant le cahier numérique www.cahiernum.net/N6KJDA

Étape 1 : Animation du personnage

1. . commencez par télécharger sur studserver les fichiers :
   - `jump_game.pyxres`, il contient les éléments du jeu (images et sons) ;
   - `jumpGame.py`, il contient un début d'implémentation du jeu.

2. Ouvrez l'éditeur de ressources de ce jeu.
   Vous remarquerez qu'il existe deux images du personnage. L'alternance de ces deux images va donner l'impression d'animation.

   Dans la fonction `draw()` , on va créer une variable `costume` :
   `costume = pyxel.frame_count//6%2`
   Rappelons que le jeu ce met à jour 60 fois par secondes. Donc `costume = pyxel.frame_count//6` s'incrémente de 1 toutes les dixième de secondes et le %2 (le reste de la division euclidienne par 2) permet à la variable `costume` de prendre alternativement les valeurs 0 et 1.

   Il suffit alors de choisir le dessin du personnage situé dans les ressources au coordonnées (0, 0) ou (0, 16) :
   `pyxel.blt(20, perso_y, 0, 0 + 16*costume ,0 , 16, 16, 12)`

3. Notre personnage marche, il lui faut maintenant sauter.
Remarquez dans la fonction `update()` l'appel à la fonction `up` : `perso_y = up(perso_y)`
C'est cette fonction qui va modifier l'ordonnée `perso_y`.

Pour simuler le déplacement du personnage (la montée quand il saute mais aussi la chute lorsqu'il retombe) nous allons utiliser la variable globale `vitesse_verticale`.
Pour l'instant le personnage ne bouge pas car `vitesse_verticale` vaut `0`.

Vous pouvez donc modifier cette variable dans la fonction `up` avec l'utilisation de la touche espace :

```
if pyxel.btnp(pyxel.KEY_SPACE) :
        vitesse_verticale = -7
```

Évidemment un problème se pose : le personnage s'envole sans espoir de redescendre. Soyez sympa avec lui, ajoutez :

```
else :
        vitesse_verticale = 0
```

4. Vous l'avez peut-être remarqué, dans la vie réelle, lorsque nous sommes dans le vide ... on tombe. Ce n'est pas le cas de notre personnage.
Votre professeur de physique vous a également appris que la vitesse de chute n'est pas constante. Pour faire simple, plus la chute est longue, plus on va vite.
Pour simuler cela, il suffit qu'à chaque appel de la fonction `up`, si le personnage ne saute pas, la `vitesse_verticale` soit décrémentée de 1.
`vitesse_verticale = vitesse_verticale + 1` au lieu de `vitesse_verticale = 0`.

Problème : même sur la terre ferme, notre personnage tombe !

A vous de jouer

Ajouter une condition pour que lorsque l'ordonnée du personnage est supérieure ou égal à 84, la `vitesse_verticale` reste constante égale à 0 (et l'ordonnée du personnage revienne à 84).

5. Un dernier problème a régler : notre personnage ne devrait pas pouvoir sauter lorsqu'il est en l'air mais uniquement lorsque ses pieds sont au sol.

A vous de jouer

Je vous laisse trouver une solution à ce problème.

Vous trouverez sur studserver une démonstration de ce à quoi devrait ressembler votre jeu à la fin de cette étape.

Étape 2 : Gestion des décors

1. Pour donner l'impression que notre personnage se déplace, nous faisons défiler les montagnes de la droite vers la gauche.
   Remarquez la présence de la ligne `montagne_x = montagne_x - 1` dans la fonction `update()`. Cette variable définit l'abscisse de l'image de la montagne, abscisse décrémentée de 1 à chaque frame.

   Dans la fonction `draw()` nous retrouvons alors la ligne suivante qui nécessite quelques explications :
   `pyxel.blt(montagne_x%320 - 160 , 40, 0, 0, 64, 160, 56)`

   Cela permet de dessiner la montagne. `montagne_x%320` est le reste de la division euclidienne de `montagne_x` par 320 c'est à dire un nombre entre 0 et 319 et donc `montagne_x%320 - 160` permet donc à l'abscisse de l'image de prendre les valeurs de -160 à 159 (c'est à dire que lorsqu'elle arrive à -160, à gauche de la fenêtre, elle repart à 159, à droite de la fenêtre).

   Évidemment cela laisse un noir entre deux images de la montagne. Pour éviter cela décommentez la ligne suivante.
   `pyxel.blt((montagne_x-160)%320-160 , 40, 0, 0, 64, 160, 56)`
   Cela permet de dessiner une autre image de la montagne mais en la décalant de 160 pixels par rapport à la première

Pour placer une image à l'écran on utilise la syntaxe :
`pyxel.blt(x, y, img, u, v, w, h, c)`
Cette instruction permet de placer à la position `(x, y)` la région dont le coin supérieur gauche est en `(u, v)` dans la banque d'images `img` et de dimensions `w, h`.
Si une valeur négative est mise pour `w` (ou `h`), la copie sera inversée horizontalement (ou verticalement). Si `c` est spécifiée, elle sera traitée comme une couleur transparente.

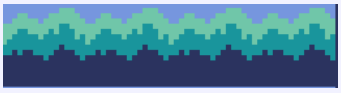
A vous de jouer

Inspirez vous de cette méthode pour faire défiler les nuages en haut de l'écran.
Pour obtenir un effet de profondeur, les nuages doivent défiler plus rapidement que la montagne.

On pourra utiliser la syntaxe
`pyxel.blt(0, 0, 0, 0, 120, 160, 16)` pour ajouter une bande de ciel bleu en haut de la fenêtre.

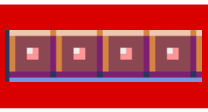
A vous de jouer

Inspirez vous de cette méthode pour faire défiler l'herbe en bas de l'écran.
Pour obtenir un effet de profondeur, l'herbe doit défiler plus rapidement que les nuages.



Vous trouverez sur studserver une démonstration de ce à quoi devrait ressembler votre jeu à la fin de cette étape.

Étape 3 : Les passerelles



Nous allons maintenant créer des passerelles de façon (presque) aléatoire sur lesquelles le personnage va pouvoir se déplacer.
Je vous encourage à commencer par regarder sur studserver ce à quoi devrait ressembler votre jeu à la fin de cette étape.

1. Nous allons reprendre la technique apprise dans le tuto 1 (pour faire apparaître les aliens).
Commençons par créer deux variables globales :

- `liste_passerelles = [[100, 90]]` qui stockera toutes les passerelles du jeu. Chaque passerelle sera stockée sous forme d'une liste à deux éléments : son abscisse et son ordonnée.
- `ordonnee_passerelle = 90` qui stockera l'ordonnée de la dernière passerelle crée (la première étant à 90)

Dans la fonction `update()`,
on appellera la fonction `creation_passerelle(liste_passerelles, ordonnee_passerelle)` avec la ligne :
`liste_passerelles, ordonnee_passerelle = creation_passerelle(liste_passerelles, ordonnee_passerelle)`

La fonction `creation_passerelle` devra toutes les 20 frames :

- choisir un nombre aléatoire 0, 1 ou 2 : `alea = randint(0, 2)` . N'oubliez pas d'importer la fonction `randint` du module `random`.
- si ce nombre est 0 (on ajoute une passerelle au dessus de la précédente si c'est possible) :
  - si `ordonnee_passerelle` est différent de 90 on modifie la valeur de `ordonnee_passerelle = ordonnee_passerelle + 30`
  - Dans tous les cas, on ajoute la passerelle `[160, ordonnee_passerelle]` à `liste_passerelles`
- si ce nombre est 1 (on ajoute une passerelle au dessus et une au dessous de la précédente ci cela est possible) on ne modifie pas `ordonnee_passerelle` mais :
  - si `ordonnee_passserelle` est différent de 90,
    on ajoute la passerelle `[160, ordonnee_passerelle + 30]`
  - si `ordonnee_passserelle` est différent de 30,
    on ajoute la passerelle `[160, ordonnee_passerelle - 30]`
- si ce nombre est 2 (on ajoute une passerelle au dessous de la précédente si c'est possible)
  - si `ordonnee_passerelle` est différent de 30 on modifie la valeur de `ordonnee_passerelle = ordonnee_passerelle - 30`
  - Dans tous les cas, on ajoute la passerelle `[160, ordonnee_passerelle]` à `liste_passerelles`
- on retourne `liste_passerelles` et `ordonnee_passerelle`

> Vous pouvez ajouter une ligne `print(liste_passerelles)` dans la fonction `update()` pour vérifier le bon fonctionnement de la fonction `creation_passerelle`.
> Si vous utilisez pyxel studio, il est possible que la console ne puisse pas afficher la liste des passerelles dès que celle ci devient trop grande et que le programme plante. Ne vous inquiétez pas ce problème sera résolu dans la suite.

2. Dessinons les passerelles.
Dans la fonction `draw()` :

```
for passerelle in liste_passerelles :
        pyxel.blt(passerelle[0], passerelle[1], 0, 0, 16, 32, 8 )
```

On ne voit pas de passerelle ? Normal, elle sont à droite de la fenêtre puisque `passerelle[0]` est toujours égal à 160.

3. Implémentons maintenant le déplacement des passerelles :
Dans la fonction `update()` :
`liste_passerelles = deplacement_passerelles(liste_passerelles)`

A vous de jouer

A vous de créer la fonction `deplacement_passerelles` pour que celles-ci se déplacent à la même vitesse que l'herbe.
Une passerelle dont l'abscisse est inférieure à -32 doit être retirée de la liste `liste_passerelles`.

Pour supprimer une passerelle, on pourra utiliser la syntaxe `liste_passerelles.remove(passerelle)`.

4. Il faut maintenant que notre personnage ne tombe pas lorsqu'il est sur une passerelle.
On va ajouter la variable `liste_passerelles` dans les arguments de la fonction `up()` :
`def up(perso_y, liste_passerelles):`

et dans la fonction `update()` :
`perso_y = up(perso_y, liste_passerelles)`

Ajoutons une variable `tombe` dans la fonction `up`, initialisée à `True` (par défaut notre personnage tombe)
Il faut maintenant modifier la fonction `up` de la façon suivante :

- si `perso_y` est supérieure ou égale à 84, tombe passe à `False` et `perso_y` devient 84
- si le personnage se situe sur une passerelle, `tombe` passe à `False` et `perso_y` prend la valeur `passerelle[1] - 16` :

```
for passerelle in liste_passerelles:
        if 30 > passerelle[0] and  -12 < passerelle[0] \
        and perso_y + 16 > passerelle[1] - 5 \
        and perso_y + 16 < passerelle[1] + 5 :
            tombe = False
            perso_y = passerelle[1] - 16
```

- si on active la touche espace et que `vitesse_verticale` est égale à 0, alors `vitesse_verticale` passe à -7
- sinon et si le personnage ne tombe pas, `vitesse_verticale` passe à 0
- sinon `vitesse_verticale` est incrémentée de 1

Vous trouverez sur studserver une démonstration de ce à quoi devrait ressembler votre jeu à la fin de cette étape.

Étape 4 : Les cerises

1. Certaines passerelles vont contenir des cerises. Pour implémenter cela on va modifier un peu la structures des passerelles en ajoutant un booléen (`True` pour une cerise, `False` s'il n'y en a pas).
   Une passerelle est maintenant une liste à trois éléments : `[ordonnée, abscisse, booléen]`.
   Lors de la création de la passerelle, on choisit un nombre au hasard entre 0 et 4, si ce nombre vaut 4, la passerelle contiendra une cerise sinon elle n'en contiendra pas.

   A vous de jouer

   Il faut maintenant dessiner la cerise sur la passerelle lorsqu'elle existe.

2. Pour supprimer la cerise lorsque le personnage entre en contact avec elle, il suffit d'ajouter la ligne `passerelle[2] = False` au bon endroit dans la fonction `up()` .

Vous trouverez sur studserver une démonstration de ce à quoi devrait ressembler votre jeu à la fin de cette étape.

Étape 5 : Gestion du score et fin de partie

A vous de jouer

A chaque cerise ramassée, le personnage gagne 1 point.
Au bout de 4 cerises qui sortent du jeu sans avoir été ramassées par le personnage, le jeu s'arrête et un message apparaît indiquant le score du joueur.

Pour afficher le score, on pourra utiliser la syntaxe
`pyxel.text(50, 64, 'GAME OVER\n score :' + str(score), 7)`

Vous trouverez sur studserver une démonstration de ce à quoi devrait ressembler votre jeu à la fin de cette étape.

Pour améliorer le jeu, on peut imaginer plein de choses :
- augmenter la vitesse au fur et à mesure ;
- Ajouter des objets à ramasser ;
- Faire perdre lorsqu'on touche le sol ...
- ... Bref faîtes preuve d'imagination, tout ou presque est possible.

Vous trouverez toute la documentation de pyxel en suivant ce lien :
https://github.com/kitao/pyxel/blob/main//docs/README.fr.md