CODING DAY

13 mai 2017

Table des matières

En quelques mots

Bienvenue à cette session du Coding Day.
Aujourd'hui, vous allez (re)découvrir la programmation à travers le langage Python, un langage orienté objet.

A travers ce sujet, vous allez découvrir les possibilités offertes par Python et son module graphique Pygame.

Ensuite, nous allons vous accompagner pour vous aider à créer la base de votre propre jeu étape par étape.

J'espère que vous prendrez autant de plaisir à réaliser votre jeu que nous avons pris de plaisir à réaliser ce sujet.

Bon courage à tous !

1. Pygame : quelques notions

1.1. Pygame, c'est quoi ?

Si vous vous intéressez à Pygame, c'est que vous avez à peu près compris les mécanismes du Python.

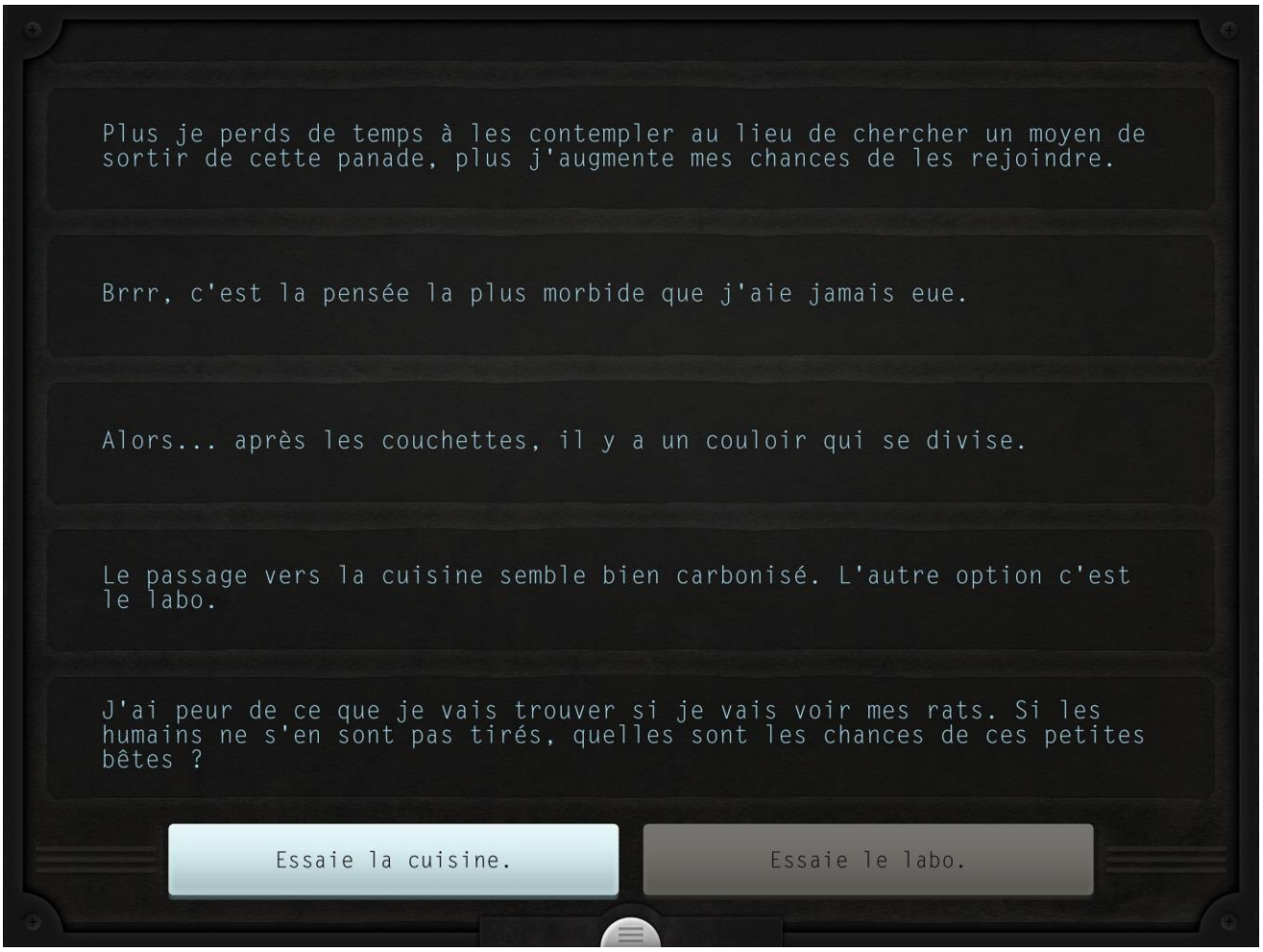
Pygame est une interface graphique pour Python basée sur la SDL (Simple Directmedia Library). Voici quelques-unes de ses utilités :

- L'affichage vidéo 2D
- La gestion de l'audio
- La gestion de périphériques de commandes (clavier, souris...)

1.2. Vous avez dit interface graphique ?

Dans la plupart des langages de programmation, on commence par coder des programmes qui se lanceront en mode console. Ainsi, il n'est pas nécessaire de se préoccuper de l'affichage ou même du son.

1.3. Un exemple vaut mieux que 1000 explications



L'image ci-dessus est tirée de Lifeline, un jeu disponible sur smartphone dont l'intégralité du contenu est en mode console. Vous devrez régulièrement faire un choix parmi plusieurs propositions pour faire avancer le scénario.

Passons maintenant à un jeu disposant d'une interface graphique.

Cette image vous semble familière ? C’est normal si vous n’êtes pas resté prisonnier sur une île déserte ces 10 dernières années. Ceci est une illustration de l’adaptation du célèbre jeu Pokemon en SDL par un fan de la saga.

1.4. Différences de conception

Si vous lisez ces lignes, c’est que vous avez survécu aux précédentes étapes de compréhension du Python voire de Pygame.

Jusqu'à maintenant, vous avez codé des scripts qui s'exécutait au fur et à mesure du fichier.

Par exemple, lors du Morpion, le programme attend qu’un joueur choisisse une case et joue pour que le joueur suivant ou l’IA joue à son tour jusqu’à ce que la partie se termine etc... Vous suivez ? :)

Pour une interface graphique, c'est différent. Ce sont désormais les évènements tels que la pression d’une touche du clavier ou de la souris qui va guider l’exécution du programme.

A titre d’exemple, prenons l’image suivante tirée d’un célèbre jeu vidéo

Imaginez votre personnage dans cette situation. Si vous n’appuyez sur aucune touche, votre programme ne va pas avancer. Par contre, vous pouvez appuyer sur Haut, Bas, Gauche et Droite pour déplacer votre personnage, voire d’autres touches pour d’autres actions.

L’exécution de votre programme s’exécute de la façon suivante :

Affichage du personnage dans son environnement

Attente de la pression d'une touche

Déplacement correspondant

...

Fin du jeu / programme.

En résumé, votre programme attend que vous lui donniez les ordres adéquates pour se poursuivre.

2. Pygame : Place à la pratique

Avant d'aller plus loin, assurez-vous que ***Python3*** et ***Pygame 1.9*** soient bien installés sur votre espace de travail. Si vous n'êtes pas sûrs, n'hésitez pas à demander autour de vous.

2.1. Première fenêtre

Nous allons désormais rentrer dans le vif du sujet. Ouvrez un fichier python avec votre éditeur préféré en ajoutant « .py » à la fin du nom que vous lui donnerez. Pour plus de lisibilité, nous utiliserons par défaut le nom « codingday.py ».

A chaque ligne où vous trouverez le caractère « # », ce qui sera suivi sera appelé commentaire et ne sera pas interprété par votre programme.

Commençons simplement.

Pour afficher une fenêtre, nous importons Pygame dans son intégralité afin de ne pas nous restreindre dans les fonctionnalités utilisées par la suite, puis nous l'initialisons afin de pouvoir poursuivre la création de la fenêtre.

```
#!/usr/bin/python3

import pygame #import de Pygame
pygame.init() #initialisation de Pygame
```

Déclarons notre fenêtre que nous appelerons « window » et donnons-lui une résolution :

```
window = pygame.display.set_mode((640,480)) # on créé notre fenêtre en 640 x 480
```

Nous faisons appel à la fonction set_mode() contenue dans le module « display » de Pygame. C'est grâce à cette fonction que nous définissons la résolution de notre fenêtre mais attention à ne pas oublier les parenthèses !

Vous pouvez déjà tester votre petit programme en l'exécutant. De notre côté, nous l'exécutons de la façon suivante :

./codingday.py

Déçu de ne voir qu'une fenêtre noire disparaître aussitôt ? C'est normal, on s'occupera de ça par la suite.

Tout d'abord, on va « customiser » un peu tout ça.

Créons une variable « title » pour donner un nom à notre fenêtre / jeu.

Créons aussi une variable « intro » qui va charger une image situé dans le dossier « images ».

```
title = "Zelda Pygame" # on nomme la fenêtre de notre jeu

intro = pygame.image.load("images/intro.jpg").convert() # on charge notre première image


pygame.display.set_caption(title)

window.blit(intro, (0,0))

pygame.display.flip()
```

window.blit va afficher l'image précédemment chargé dans la variable intro alors que la fonction display.flip() va nous servir à mettre à jour (« rafraichir ») votre fenêtre.

Si vous n'êtes pas encore perdu, votre programme devrait ressembler à ça pour le moment.

```
#!/usr/bin/python3


import pygame # permet d'utiliser pygame

pygame.init() # on initialise pygame


title = "Zelda Pygame" # on nomme la fenetre de notre jeu

window = pygame.display.set_mode((640,480)) # on cree notre fenetre en 640 x 480

intro = pygame.image.load("images/intro.jpg").convert() # on charge notre premiere image


pygame.display.set_caption(title)

window.blit(intro, (0,0))

pygame.display.flip()
```

En lançant votre programme, vous devriez constater que le fond noir initial à laisser place à une image.

Néanmoins, l'image disparait toujours car votre programme estime qu'il a été au bout de ce qu'il devait exécuter.

2.2. Boucle infinie

Pour maintenir l'affichage de la fenêtre, nous allons avoir besoin d'utiliser une boucle. Pour cela, nous créons une nouvelle variable appelée « loop » à laquelle nous allons donner la valeur de « 1 ».

Avec le mot clé « while », nous allons « faire boucler » notre programme jusqu'à ce que la valeur de « loop » ne soit plus égale à 1.

```
#!/usr/bin/python3

import pygame # permet d'utiliser pygame

pygame.init() # on initialise pygame


title = "Zelda Pygame" # on nomme la fenetre de notre jeu

window = pygame.display.set_mode((640,480)) # on cree notre fenetre en 640 x 480


loop = 1

while loop:

    intro = pygame.image.load("images/intro.jpg").convert() # on charge notre premiere image

    pygame.display.set_caption(title)

    window.blit(intro, (0,0))

    pygame.display.flip()
```

Vous avez compris ? Félicitations, vous venez de réaliser votre première boucle du programme. Votre image devrait être maintenue si vous relancez votre programme.

2.3. Apprenons à structurer notre code

Maintenant que notre premier objectif est accompli, nous allons préparer le terrain pour la suite. Nous allons créer des variables qui vont en quelque sorte stocker les images ou animations généralement nommées « sprites ».

```
#!/usr/bin/python3


import pygame # permet d'utiliser pygame


################ Gestion des sprites ###############


title = "Zelda Pygame" # on nomme la fenetre de notre jeu
image_icone = "images/link_right.gif"
image_intro = "images/intro.jpg"
image_back = "images/back.jpg"
image_wall = "images/wall.gif"
image_start = "images/thumb_Hole.gif"
image_triforce = "images/triforce.gif"


nb_sprite = 15
sprite_size = 30
window_size = nb_sprite * sprite_size
```

Comme vous êtes des pros du Python désormais, vous allez définir la résolution de votre fenêtre en vous appuyant sur un nombre de cases défini par le nombre de sprites souhaité et la taille de chacune des sprites.

C'est confus ? Prenons l'exemple ci-dessus. Chaque sprite de la fenêtre fera 30 pixels de hauteur et 30 pixels de largeur. La fenêtre elle-même sera constituée de 15 sprites de hauteur et 15 pixels de largeur.

```
################ Boucle de jeu ####################


pygame.init() # on initialise pygame
window = pygame.display.set_mode((window_size, window_size))
icone = pygame.image.load(image_icone)
```

Modifions notre variable window afin que la résolution de notre fenêtre réponde à nos besoins.

2.4. Evènements clavier

Reprenons l'avancée de notre programme. Nous allons maintenant découvrir les évènements clavier. En quoi cela consiste ?

Lorsque vous cliquez sur une touche de votre clavier, vous interagissez avec votre machine. La pression de la touche est considérée comme un évènement clavier au même titre qu'un clic de souris est considéré comme… un évènement souris ☺

Retournons dans notre boucle de jeu précédemment réalisée qui permettait de maintenir l'image affichée à l'écran.

```
################ Boucle de jeu ####################
loop = 1
while loop:
        intro = pygame.image.load(image_intro).convert()
        window.blit(intro, (0,0))
        pygame.display.flip()

        loop_intro = 1
        loop_game = 1
```

Quoi de nouveau ici ? Nous avons changé quelque peu la variable « intro » pour qu'elle charge l'image directement stockée dans la variable « image_intro » vue dans la partie précédente.

Nous avons ajouté 2 loop qui vont avoir leur importance.

Comme pour la première boucle, nous allons faire appel au mot clé « while ». Ainsi, tant que la valeur de « loop_intro » sera égale à 1, votre programme attendra et pourra interpréter la pression de certaines touches clavier via le module « KEYDOWN » (pression d'une touche).

```
        while loop_intro:
                for event in pygame.event.get():
                        if event.type == pygame.QUIT or event.type
 == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
                                loop_intro = 0
                                loop_game = 0
                                loop = 0
```

On lance la boucle infinie avec « while » puis une boucle « for », qui parcourt tous les événements reçus grâce à la fonction get() du module « event » de Pygame. Je sens que votre cerveau commence à chauffer.

Cette condition va vérifier si vous lancez un évènement de fermeture de votre programme en cliquant sur la croix ou pressant la touche « échap » de votre clavier.

Si l'une de ces conditions est remplie, on change la valeur de nos « loop » ce qui va provoquer la sortie du programme.

2.5. Mise en place du jeu

Nous venons de voir comment quitter notre programme en appuyant sur la touche « échap » du clavier. Que diriez-vous de ne pas vous contenter de l'affichage de votre image et de constituer l'environnement de votre jeu ? Partant ? Allons-y !

Reprenons la précédente condition. Si nous déclenchions un évènement visant à quitter le jeu, cela entrainait la sortie du programme.

Nous pourrions désormais déclencher une autre action si nous appuyions sur une autre touche. Ajoutons tout d'abord une nouvelle variable à notre précédente condition.

```
                    if event.type == pygame.QUIT or event.type
== pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
```

```
loop_intro = 0
loop_game = 0
loop = 0
load_map = 0
```

Pourquoi ajouter une variable load_map alors que nous n'avons pas encore généré ni évoquer la map ? Patience… ☺

Ajoutons une condition avec un nouveau mot clé « elif ». Dans le cas où nous n'appuyons pas sur « échap » mais sur la touche « entrée » par exemple, nous pourrions déclencher une autre action.

```
elif event.type == pygame.KEYDOWN:
    if event.key == pygame.K_RETURN:
        loop_intro = 0
        load_map = 'LinkToThePast'
```

Rien ne se passe en appuyant sur « entrée » ? Pas de panique.

2.6. Prise en charge de la map : un peu d'objet

Nous allons maintenant créer notre map qui va nous servir de paramètre de jeu.

```
#!/usr/bin/python3
import pygame
################ Gestion des sprites ###############
…
################ Generation de la map ##############
class Map:
        def __init__(self, file):
                self.file = file
                self.structure = 0
```

Voici une notion qui vous est probablement inconnue : le mot clé « class ». Une classe est une sorte de conteneur où sont créés les objets. En programmation, une classe regroupe diverses fonctions et variables. Ici, nous créons donc une classe « Map » qui va contenir tous les éléments essentiels à son bon usage.

« __init__ » va servir à initialiser / créer votre classe. Elle aura besoin d'un fichier « file » par la suite dont elle récupérera les informations.

Nous allons maintenant créer une fonction qui va afficher divers composants de notre map en chargeant les images adéquates.

```
        def show(self, window):
```

```
                wall = pygame.image.load(image_wall).convert()
                start = pygame.image.load(image_start).convert()
                triforce = pygame.image.load(image_triforce).convert_alp
ha()
```

A priori, rien de compliqué ici, vous devriez être apte à comprendre les lignes ci-dessus ☺

Pimentons un peu la chose. Revenons dans notre boucle de jeu.

Nous avons auparavant créé une variable load_map qui va (enfin) nous être utile.

Dans le cas où la map adressé à « load_map » a été « trouvé », nous allons pouvoir générer son affichage.

Nous allons créer de nouvelles variables pour arriver à nos fins.

```
if load_map != 0:
        back = pygame.image.load(image_back).convert()
        map = Map(load_map)
        map.show(window)


        window.blit(back, (0,0))
        map.show(window)
        pygame.display.flip()
```

- « back » va charger l'image de fond de notre map

- « map » va charger le fichier de configuration contenant des éléments tels que votre futur personnage, obstacles et autres éléments selon votre convenance.
- « map.show » fait appel à la fonction « show » évoquée juste avant pour (enfin) générer son affichage.

Vous devriez être familier avec les 3 dernières lignes déjà rencontrées lors de l'affichage de votre première fenêtre.

Si tout se passe bien, vous devriez enfin visualiser la map après avoir appuyé sur « entrée » après votre première image.

Prenez le temps de manipuler votre code avant de passer à la suite. C'est en pratiquant et en faisant des tests que vous comprendrez le fonctionnement logique de votre programme.

Pour que votre programme soit plus agréable à utiliser (à quitter surtout), je vous suggère d'ajouter les évènements permettant de quitter une fenêtre à votre programme afin de quitter votre map fraichement affichée proprement. ☺

2.7. Création de votre personnage

Que serait un jeu sans personnage ? Un jeu sans personnage évidemment. Par chance, nous allons maintenant créer le personnage de vos rêves… ou presque !

```
#!/usr/bin/python3
import pygame # permet d'utiliser pygame
################ Gestion des sprites ###############
…
```

```
################ Generation de la map ##############
…
################ Creation du personnage ############
class Perso:
        def __init__(self, right, left, up, down, map):
                self.right = pygame.image.load(right).convert_alpha()
                self.left = pygame.image.load(left).convert_alpha()
                self.up = pygame.image.load(up).convert_alpha()
                self.down = pygame.image.load(down).convert_alpha()

                self.case_x = 0 #on positionne le personnage créé
                self.case_y = 0
                self.x = 0
                self.y = 0

                self.direction = self.down
```

Rien de bien nouveau ici. Nous venons de créer une classe Perso qui va contenir les différents mouvements possibles de celui-ci et les sprites (images) adaptées à chacun de ses déplacements.

Enfin, on a défini la position de départ du personnage.

Afin de finaliser l'affichage du personnage et la mise à jour de l'image de ce dernier lors de ses déplacements, 2 nouvelles lignes ont été soigneusement distillées, saurez-vous les retrouver ? ☺

```
        if load_map != 0:
                back = pygame.image.load(image_back).convert()
                map = Map(load_map)
                map.show(window)
                link = Perso("images/link_right.gif", "images/link_left.gif",
                        "images/link_up.gif", "images/link_down.gif", map)
```

```
                window.blit(back, (0,0))
                map.show(window)
                window.blit(link.direction, (link.x, link.y))
                pygame.display.flip()
```

2.8. Le plus difficile pour (presque) finir.

Nous avions évoqué auparavant que nous avions créé la map. C'est en partie vrai. Nous avions en effet initialisé une map mais il nous manque certains éléments pour que notre map soit la plus complète possible.

Cette partie étant un peu plus complexe, nous allons gracieusement vous offrir les lignes suivantes sans que vous n'ayez trop à vous attarder dessus. Retournons dans la classe « Map ».

```
def create(self):
                with open(self.file, "r") as file:
                        structure_map = []
                        for line in file:
                                map_line = []
```

```
                    for sprite in line:
                        map_line.append(sprite)
                    structure_map.append(map_line)
                self.structure = structure_map
```

Pour faire « simple », nous ouvrons ici un fichier qui est le fichier contenant les informations de votre map. Ces informations sont analysées et stockées.

Revenons dans la fonction show et ajoutons-y les lignes suivantes :

```
    def show(self, window):
    …
        nb_line = 0
        for line in self.structure:
            nb_pos = 0
            for sprite in line:
                x = nb_pos * sprite_size
                y = nb_line * sprite_size
                if sprite == 'W':
                    window.blit(wall, (x,y))
                elif sprite == 'L':
                    window.blit(start, (x,y))
                elif sprite == 'T':
                    window.blit(triforce, (x,y))
                nb_pos = nb_pos + 1
            nb_line = nb_line + 1
```

Avec les informations du fichier de la map précédemment stockées, nous définissons ici différents éléments qui seront affichés et « gérés » par sur la map. Nous arrivons à la dernière étape consacrée aux déplacements du personnage en considérant ces éléments de la map.

2.9. Déplacements et collisions

Retournons dans la classe « Perso ». Nous allons maintenant créer la fonction « move » qui, comme son nom l'indique va permettre le déplacement de votre personnage.

```
        def move(self, direction):
                if direction == 'right':
                        if self.case_x < (nb_sprite - 1):
                                if self.map.structure[self.case_y][self.case_
x + 1] != 'W':
                                        self.case_x = self.case_x + 1
                                        self.x = self.case_x * sprite_size
                        self.direction = self.right
```

Pour générer le déplacement, nous considérons la position réelle de notre personnage en fonction de sa position par rapport aux pixels sur la fenêtre.

Les déplacements sont gérés de façon à ce que votre personnage ne puisse ni sortir de l'écran, ni « traverser » un mur.

Comme vous l'aurez judicieusement remarqué, ceci ne concerne que les déplacements à droite ! ☺ Vous devriez être aptes à créer les déplacements dans les 3 autres directions.

Pour que tout fonctionne dans le meilleur des mondes, il faut aussi lier le déplacement à la pression d'une touche de votre clavier.

Pour cette fois encore, je vous laisse rejeter un œil du côté des évent clavier. ☺

3. A vous de jouer !

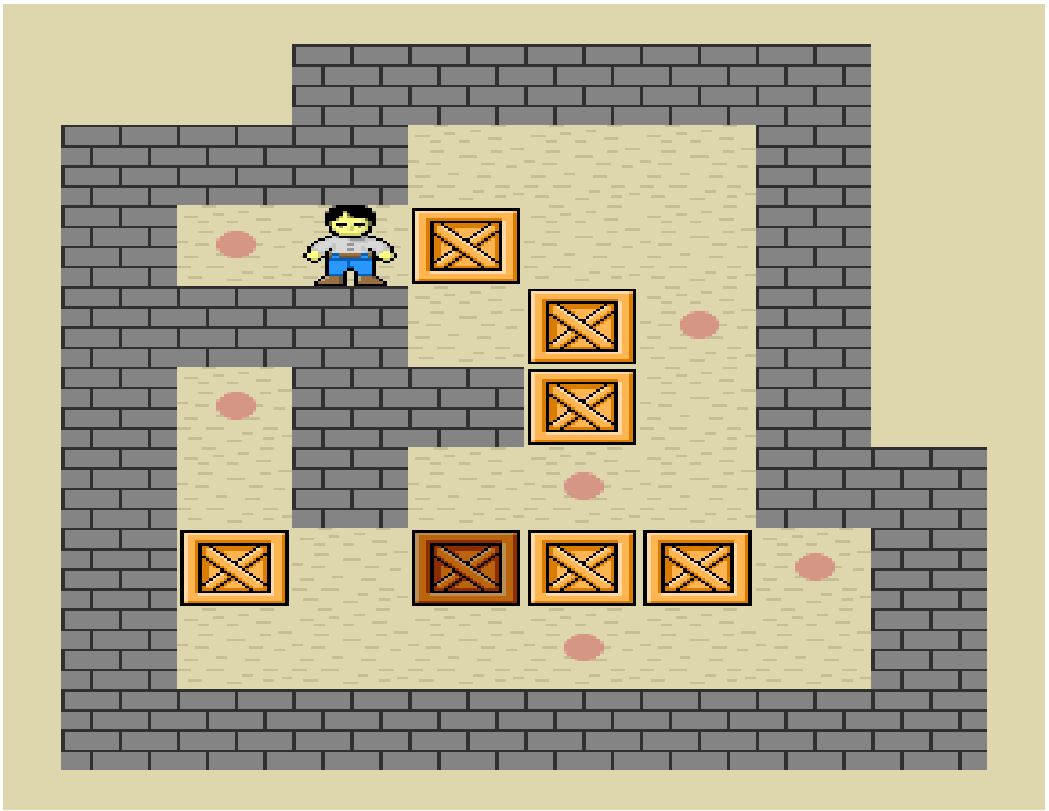
Si vous lisez ces lignes, c'est que vous avez peut-être survécu et relevé le challenge des précédentes étapes. Peut-être avez-vous même un début de jeu jouable.

Il est désormais temps d'aller plus loin. De nombreuses possibilités s'offrent à vous. Vous pouvez améliorer votre jeu en y ajoutant diverses fonctionnalités comme une ambiance sonore, un mode multi-joueurs pour les plus farouches d'entre vous, un mode de jeu à la souris etc… Les possibilités ne se limitent qu'à votre imagination !

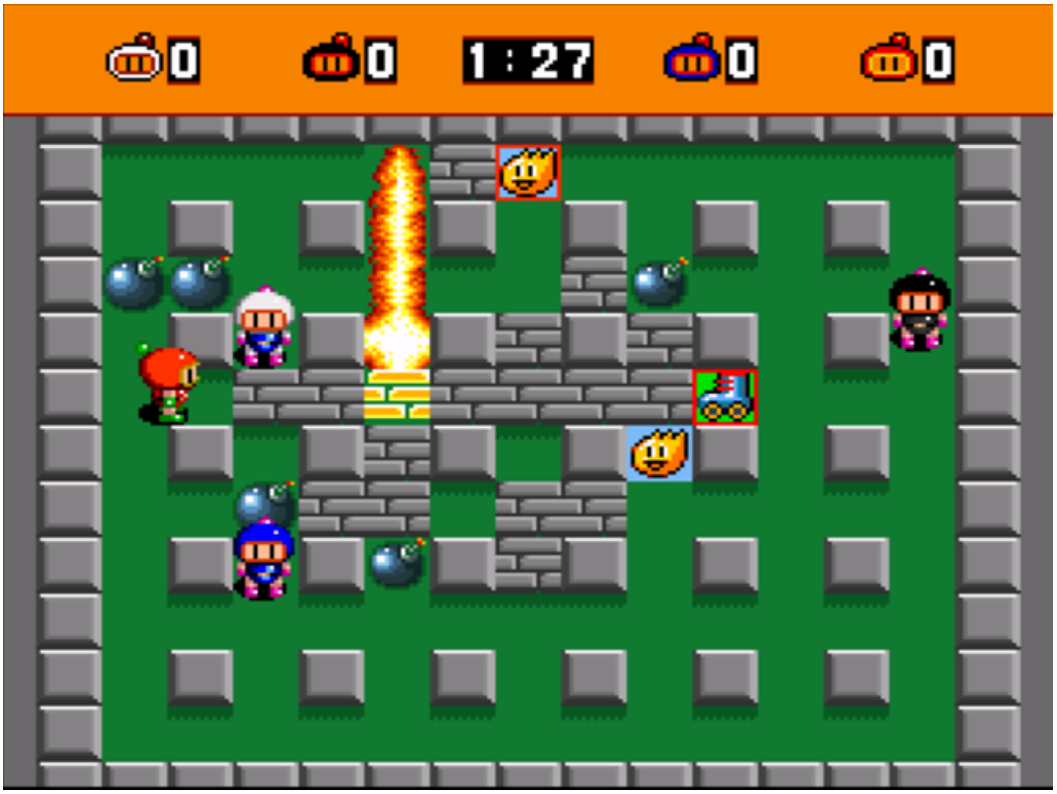
Vous pouvez aussi vous inspirer de certains monuments du jeu vidéo qui ont marqué des générations de joueurs pour donner une tournure à votre jeu.

On commence par l'incontournable Pacman.

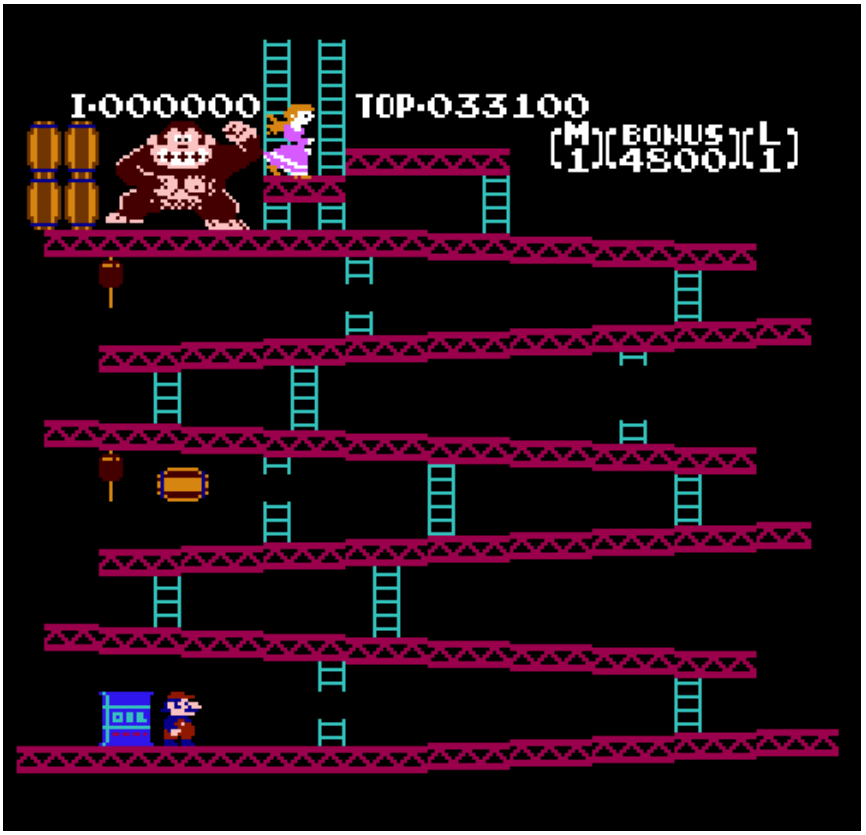
Le Sokoban est un peu moins connu de nos jours. Vous devez contrôler un personnage qui doit pousser des casses sur des emplacements spécifiques sans se retrouver bloqué.



L’indémodable Bomberman, sorti sur quasiment toutes les consoles à ce jour.



Dans un registre quelque peu différent, nous retrouvons Donkey Kong sorti sur Nes. Un jeu qui marquera l’histoire du jeu vidéo avec l’apparition de la célèbre mascotte de Nintendo.



Lâchez-vous et épatez-nous !