Projet jeu

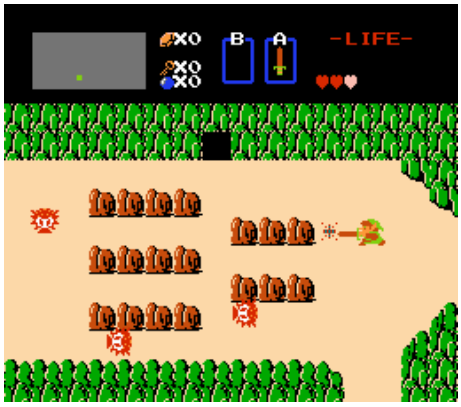
GCC ! – Prologin

2019

Introduction

Bravo, vous avez survécu jusqu'à la fin de la semaine ! Il est temps de passer aux choses sérieuses. Pour clore le stage, on vous propose de coder un petit jeu qui vous fera utiliser toutes les notions que vous avez vues jusqu'ici. Ce sujet va vous guider à travers la réalisation d'un jeu. Nous n'allons pas vous donner un jeu précis à programmer mais un ensemble de méthodes pour réaliser le jeu que vous aurez imaginé.

La première étape est donc d'avoir une idée approximative de ce que vous souhaitez réaliser. N'ayant que 2 jours pour programmer ce jeu, on se limitera a des jeux en deux dimensions. En particulier, vous choisirez un jeu avec une vue d'en haut (figure 1a) ou un jeu avec une vue de coté (figure 1b), mais si vous avez quelque chose d'autre en tête, n'hésitez pas à en parler avec un organisateur. Si vous ne pensez à rien, vous pouvez réimplémenter un jeu classique : pacman, tetris, bomberman, mario, snake, morpion



(a) Capture d'écran de *The Legend of Zelda* avec une vue d'en haut.

(b) Capture d'écran de *Super Mario Bros.* avec une vue de coté.

FIGURE 1 – Comparaison des deux points de vue classiquement rencontrés dans les jeux 2D.

Ce sujet est composé de trois parties :

— La première partie présente les bases de programmation nécessaire à la plupart des jeux, vous devez la suivre quelle que soit votre idée.

— La deuxième partie est une liste de fonctionnalités qui peuvent vous être utiles, par exemple il y a une section sur comment gérer le saut d'un personnage, mais celle-ci ne devrait vous être utile que si vous contrôlez un personnage en vue de coté.
— La troisième partie est une référence listant des fonctions disponibles dans la bibliothèque logicielle Pygame que vous allez utiliser pour implémenter votre jeu.

Si vous regardez attentivement les captures d'écran de la figure 1, vous remarquerez que des images sont répétées, par exemple le sol de la figure 1b est composé d'une même image répétée une quinzaine de fois horizontalement. Ces images de base sont appelées *sprites*, ce sont les briques de bases permettant de construire un jeu visuellement riche. Dans un premier temps, on vous demandera de vous limiter aux *sprites* que nous vous avons fournies et qui sont listées en annexe A.

Le programme que vous allez écrire va être long et il est important que vous soyez toujours capables de le comprendre demain. Par ailleurs, la gentillesse des organisateurs est proportionnelle à la lisibilité de votre code. C'est pourquoi, il vous est fortement conseillé de bien formater votre code et d'écrire pleins de petites fonctions chacune se chargeant d'une partie de la logique du jeu, par exemple on pourra avoir une fonction `afficher_carte` et une autre `gerer_deplacements`. En écrivant de grosses fonctions monstrueuses vous nuisez à la santé des gentils organisateurs.

Fondamentaux

Pour commencer, nous allons vous guider à travers l'affichage des `sprites` et comment les faire se déplacer. Pour programmer votre jeu, vous allez utiliser la bibliothèque Pygame, toutes les fonctions de cette bibliothèque commenceront donc par `pygame.`, par exemple `pygame.transform.scale`.

Structure du programme

On distinguera deux partie à votre programme : l'initialisation et la boucle de jeu. L'initialisation n'est faite qu'une seule fois au début du programme, alors que la boucle de jeu va être exécutée une centaine de fois par seconde. Par exemple, dans la partie initialisation on initialisera le score du joueur à 0 : `score = 0`, alors que dans la boucle de jeu on augmentera le score du joueur lorsque certaines conditions sont vérifiées `score += 1`. Tout au long du projet, il vous faudra faire bien attention à ne pas écrire du code trop lent dans la boucle de jeu, si vous voulez que la boucle s'exécute 100 fois par seconde, il est nécessaire qu'une itération prenne moins de 10 millisecondes.

La bibliothèque Pygame nécessite d'être initialisée, pour cela vous utiliserez la fonction `pygame.init`, vous commencerez votre programme par :

```
1 import pygame
2
3 # Initialisation
4 pygame.init()
5
6 while True:
7     # Boucle de jeu
```

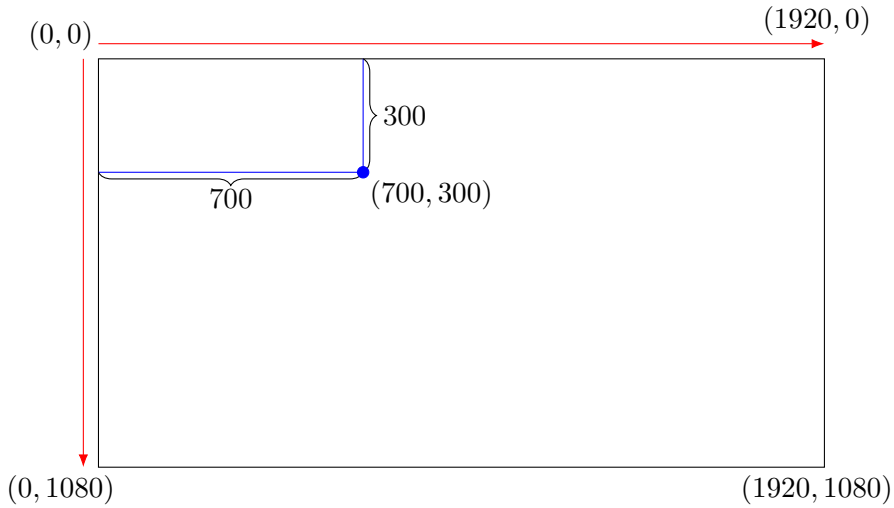



FIGURE 2 – Illustration du système de coordonnées d'un écran de 1920 pixels de largeur et 1080 pixels de hauteur. Le point $(700, 300)$ est marqué en bleu.

L'écran

Pour la plupart des programmes, l'unité d'affichage de base est le pixel : un carré de l'ordre d'un dixième de millimètre de coté qui ne peut être que d'une seule couleur donnée. L'écran peut être vu comme une grille de pixels, la particularité étant que l'origine se trouve en haut à gauche (voir figure 2).

La première étape que vous allez tester est l'ouverture d'une fenêtre. Pour ce faire vous utiliserez la fonction `pygame.display.set_mode` qui prend en argument une paire d'entiers : (largeur, hauteur), la taille de la nouvelle fenêtre. Cette fonction renvoie un objet représentant la fenêtre ainsi créée, il sera nécessaire de passer cet objet à d'autres fonctions pour pouvoir dessiner dedans, gardez le dans une variable ! Il n'est nécessaire de créer qu'une seule fenêtre au début du programme, cet appel de fonction va donc dans la partie initialisation.

La seconde fonction dont vous aurez besoin est `pygame.display.update`. Cette fonction (qui ne prend pas d'argument) va dessiner le contenu de la fenêtre à l'écran. Cette action doit être effectuée à chaque tour de boucle, votre programme va donc demander à la bibliothèque Pygame de dessiner l'écran plus d'une centaine de fois par seconde !

Votre programme devrait donc ressembler au code suivant :

```
1 import pygame
2
3 # Initialisations
4 pygame.init()
5 ecran = pygame.display.set_mode((800, 600))
6 #
7
8 while True:
9     # Boucle de jeu
```


```
10      #
11      pygame.display.update()
```

Écrivez et exécutez ce code. Pour l'instant on n'affiche rien, la fenêtre reste donc vide.

Une première fonction que vous pouvez ajouter est :

```
1   ecran.fill(pygame.Color('blue'))
```

Celle-ci se place dans la boucle de jeu et vous permet de changer la couleur de fond de la fenêtre. Pour choisir la couleur, vous pouvez écrire le nom de la couleur en anglais, une valeur hexadécimale comme en HTML (`"#8000FF"`) ou des valeurs RGB entre 0 et 255.

Rectangles

La bibliothèque Pygame fait un usage intensif d'objets de type Rectangle. Pour créer un nouveau rectangle vous pouvez utiliser `pygame.Rect` qui prend 4 arguments (x, y, w, h) où (x, y) sont les coordonnées du coin supérieur gauche du rectangle et w et h sont respectivement la largeur (*width*) et la hauteur (*height*) du rectangle. Par exemple, sur la figure 2, le rectangle en haut à gauche délimité par les traits bleus correspond au rectangle `pygame.Rect(0, 0, 700, 300)`.

De nombreux descripteurs sont disponibles pour accéder aux différents points d'intérêt d'un rectangle, lorsque vous avez défini une variable `r` comme étant un rectangle, vous pouvez accéder au point supérieur gauche avec `r.x` et `r.y`, à sa taille avec `r.w` et `r.h`, mais aussi aux coordonnées de son centre avec `r.centerx` et `r.centery`, ou encore à son coté droit avec `r.right`, son coté inférieur avec `r.bottom`, etc

Pour dessiner un rectangle coloré, Pygame met à votre disposition la fonction `pygame.draw.rect` dans la boucle de jeu, par exemple :

```
1   rectangle = pygame.Rect(100, 300, 200, 100)
2   pygame.draw.rect(ecran, pygame.Color('red'), rectangle)
```

Les rectangles sont des objets dit *mutables*, cela signifie que lorsque `r1` est un rectangle et que vous écrivez `r2 = r1`, cela ne crée pas de nouveau rectangle mais donne juste un nouveau nom au rectangle `r1` déjà existant. En particulier, faites bien attention au code suivant :

```
1   import pygame
2   r1 = pygame.Rect(2, 2, 10, 10)
3   r2 = r1
4   r1.x = 3
5   print(r2.x)
```

Celui-ci affiche 3 et non 2 comme on pourrait s'y attendre [1]. Pour effectuer une vraie copie du rectangle `r1`, il faudra remplacer la ligne 4 par `r2 = r1.copy()`.

1. Si `r1` et `r2` étaient des entiers ce code afficherait bien 2.

Images

Nous allons finalement passer à l'affichage d'images, cela se fait en deux parties : d'abord l'image est chargée en mémoire, puis elle est affichée dans une fenêtre à proprement parler. Pour charger une image, vous utiliserez l'appel suivant :

```
1 image = pygame.image.load("chemin/sprite.png").convert_alpha()
```

La fonction `pygame.image.load` prend en argument le chemin vers un fichier image et charge ce fichier en mémoire. La fonction `convert_alpha` permet quant à elle de convertir l'image dans un format rapide et optimisé pour Pygame. Ces deux fonctions sont plutôt lentes, il faut donc les appeler dans la partie initialisation, ne chargez pas d'image dans la boucle de jeu, c'est bien trop lent !

Une fonction utile sur les images est `get_rect`, celle-ci permet de récupérer un rectangle de la taille de l'image. Le rectangle ainsi retourné est ancré en $(0, 0)$ et est de la même taille que l'image.

Enfin, pour afficher l'image, il faut utiliser la fonction `blit` sur l'objet `ecran` renvoyé par la fonction `pygame.display.set_mode` vu à la section 2.2. Cette fonction prend deux arguments : l'image à afficher (que vous avez chargée avec la fonction `pygame.image.load`) et un rectangle où afficher cette image. Il faudra que ce rectangle soit de la même taille que l'image a afficher. En général, on utilise la fonction `get_rect` pour cela, on a alors un code de la forme :

```
1 rectangle = image.get_rect()
2 rectangle.x = 200
3 rectangle.y = 100
4 ecran.blit(image, rectangle)
```

Ce code qui affiche une image aux coordonnées $(200, 100)$, doit être écrit dans la boucle de jeu. Essayez d'afficher une image, si vous avez un soucis, appelez un organisateur.

Remarquez que changer la taille du rectangle (par exemple `rectangle.w`), ne suffit pas pour changer la taille de l'image. Pour ce faire, il faudra d'abord redimensionner l'image comme vu section 4.2.

Évènements

Pour communiquer avec le programme, l'utilisateur (dans ce cas là, le joueur) va envoyer des *évènements* au programme. Un événement peut prendre de nombreuses formes : c'est l'appui sur une touche du clavier ou un mouvement de la souris, mais également la fermeture d'une fenêtre, sa réduction ou son redimensionnement. Pour faire court, toute interaction avec le programme se fait à travers des évènements.

Le premier événement que vous allez capturer est la fermeture du programme. Pour fermer la fenêtre Pygame, il faut utiliser l'appel de fonction `pygame.quit()` qui est un peu l'inverse de `pygame.init()`. Puis, pour arrêter votre programme à proprement parler, nous allons utiliser l'appel de fonction `sys.exit(0)`, pour cela il est nécessaire d'ajouter `import sys` lors de l'initialisation du programme. Cet appel permet de sortir directement du programme sans avoir à attendre que toutes les instructions soient exécutées.

```
for evenement in pygame.event.get():
    if evenement.type == pygame.QUIT:
        pygame.quit()
        sys.exit(0)
```

Le traitement des évènements doit impérativement se faire dans la boucle de jeu : on va vérifier une centaine de fois par seconde si l'utilisateur a appuyé sur une touche ou, dans ce cas là, s'il a fermé le programme. La fonction `pygame.event.get` renvoie une liste des événements ayant eu lieu depuis le dernier tour de boucle. Si vous êtes curieux du type d'évènement reçu par le programme vous pouvez ajouter une ligne `print(evenement)` avant le `if` et regarder ce qui est affiché quand vous bougez la souris ou utilisez le clavier dans la fenêtre du jeu. Pour chaque évènement reçu, on vérifie si son type est `pygame.QUIT`, si un tel évènement a eu lieu, on ferme le programme.

Déplacements

Avant de regarder comment capturer les évènements correspondants à l'appui d'une touche du clavier, nous allons voir comment déplacer des images. Cela se fait très simplement : il suffit de redessiner la même image à des emplacements différents. Par exemple, initialisez une variable `posx` à 0 hors de la boucle de jeu, puis incrémentez là à chaque tour de boucle pour compter combien de tours ont été effectués jusqu'à présent. Vous pouvez maintenant modifier la position du rectangle où vous dessinez votre image pour que celle-ci dépende de `posx`. Par exemple, au lieu de faire `rectangle.x = 100`, vous pouvez faire `rectangle.x = posx`.

Essayez d'afficher une image qui se déplace en diagonale, du coin supérieur gauche vers le coin inférieur droit.

Utiliser le clavier

La capture des touches du clavier peut se faire comme vu à la section 2.5, à l'aide de la fonction `pygame.event.get` tout comme `pygame.QUIT`, mais pour chaque touche du clavier, Pygame a deux évènements : un lorsque la touche est enfoncée et un autre lorsque la touche est relachée. Traiter les évènements de cette manière est très fastidieux, pour nous simplifier la tâche, une autre fonction est disponible : `pygame.key.get_pressed`. Cette fonction renvoie un tableau avec autant de valeur que de touche sur le clavier, initialement toutes les valeurs sont 0. Lorsqu'une touche est enfoncée la case du tableau correspondante est mise à 1. Pour savoir si la touche droite est enfoncée on pourra regarder la valeur du tableau en l'indice `pygame.K_RIGHT` :

```
touches = pygame.key.get_pressed()
if touches[pygame.K_RIGHT] == 1:
    print("Touche droite enfoncée")
```

De même que `pygame.K_RIGHT` désigne la flèche de droite, il existe une variable Pygame pour chaque touche du clavier, celles-ci sont listées section 4.1.

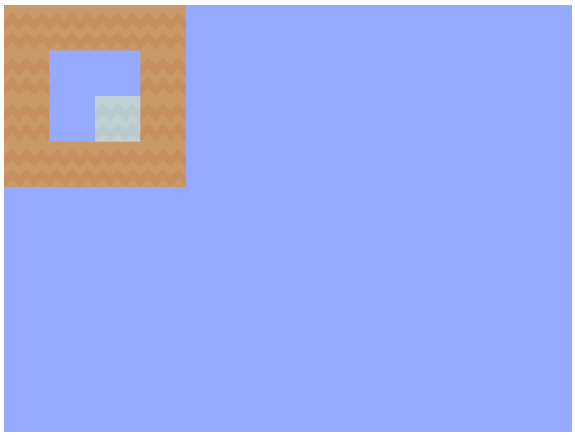


FIGURE 3 – Rendu du tableau `carte` dans une fenêtre de taille 800×600 en utilisant les *sprites* `grassCenter.png` pour `image1` et `snowCenter.png` pour `image2`.

Si vous recopiez le code ci-dessus, une ligne sera écrite dans le terminal à chaque tour de boucle lorsque la flèche droite est enfoncée. En vous inspirant de la section 2.6, modifiez votre programme pour pouvoir déplacer une image à gauche et à droite à l'aide des flèches directionnelles.

Carte d'un niveau

Écrire autant de `blit` que d'image à l'écran est beaucoup trop long et fastidieux. On préfèrera utiliser un tableau représentant la carte du jeu. Pour cela, chargez deux images : `image1` et `image2` et créez un tableau comme suit :

```
1 carte = [[1, 1, 1, 1],
2          [1, 0, 0, 1],
3          [1, 0, 2, 1],
4          [1, 1, 1, 1]]
```

Vous pouvez alors itérer sur le tableau `carte` avec deux boucles `for` imbriquées. Lorsque vous rencontrez un 1, faites `blit` d'`image1`, lorsque vous rencontrez un 2, faites un `blit` d'`image2` et lorsque vous rencontrez un 0, ne faites rien. Toute la difficulté réside en savoir où `blit` ces images. En effet la fonction `blit` prend le rectangle où dessiner l'image en paramètre. Il faudra donc faire dépendre les coordonnées de ce rectangle des indices des deux boucles `for`. Le résultat que vous devriez obtenir est donné figure 3.

Gestion des collisions

Enfin, si vous avez implémenté les sections 2.7 et 2.8, vous avez dû remarquer qu'il est possible d'utiliser les touches directionnelles pour déplacer l'image n'importe où, y compris hors de la carte. Pour empêcher les mouvements hors de la carte, vous pouvez commencer par limiter la position de l'image lorsque celle-ci est déplacée : si une coordonnée passe en dessous de zéro, celle-ci est laissée à zéro à la place.

La collision avec d'autres images est plus difficile à gérer. Il est nécessaire de commencer par stocker une liste des rectangles solides qui ne pourront pas être traversés. Un bon endroit pour faire cela est dans la boucle de dessin de la carte : on commence avec une

liste vide `solides = []`, puis pour chaque `i`, `j`, si `carte[i][j]` représente un objet solide, le rectangle correspondant est ajouté à `solides`.

Lorsqu'une touche est pressée, on va mettre à jour les coordonnées d'une image à condition que le nouveau rectangle n'a pas d'intersection avec un rectangle de la liste `solides`. Pour tester s'il existe une intersection entre deux rectangles `r1` et `r2`, Pygame nous fournit la fonction :

```
collision = r1.colliderect(r2)
if collision:
    # ne pas se déplacer
```

Où `collision` est un booléen égal à `True` s'il existe une collision, `False` sinon.

Avec la technique décrite ci-dessus vous remarquerez que l'image ne s'approche pas totalement des murs comme on s'y attendrait. Pour pallier à cela, une fois que vous avez trouvé le rectangle solide avec lequel vous êtes en collision, vous pouvez effectuer un mouvement de la forme :

```
personnage.left = solide.right
```

Cette ligne permet de déplacer la gauche du personnage de telle manière à ce que ce soit exactement la droite du solide. Cette action est donc à effectuer si on trouve une collision à gauche, c'est à dire lorsque l'on se déplace vers la gauche et qu'on rencontre un obstacle. Une code similaire devra être écrit pour toutes les directions (`left`, `right`, `top`, `bottom`).

Finalement, pour gérer correctement les mouvements diagonaux, il vous est fortement conseillé de séparer votre détection de collision en deux parties :

- Essayer de se déplacer horizontalement
- Résoudre les collisions horizontales
- Essayer de se déplacer verticalement
- Résoudre les collisions verticales

Cet ordre d'action vous assure que vos images ne se bloqueront pas dans les coins des solides de la carte.

Pour aller plus loin

Cette section présente plusieurs fonctionnalités qui ne vous seront pas nécessairement utiles, vous pouvez directement regarder les parties qui vous intéressent.

Écrire du texte

Pour écrire un texte à l'écran, il faut commencer par choisir la police d'écriture à l'aide de la fonction `pygame.font.Font`, cette fonction prend deux arguments : le nom de la police d'écriture et sa taille. Il est aussi possible de remplacer le nom de la police par `None` pour laisser la police par défaut.

À partir d'une police d'écriture, il nous est possible de générer du texte avec `police.render` qui prend trois arguments : le texte à afficher, antialiasing [2] et la couleur du texte. Après

2. Qu'on laissera toujours à `True`.

cela, afficher du texte se fait comme afficher une image, avec `ecran.blit`. On obtient alors un code de la forme :

```
police = pygame.font.Font(None, 50)
texte = police.render("Mon texte", True, pygame.Color('green'))
rectangle = texte.get_rect()
ecran.blit(texte, rectangle)
```

Utiliser la souris

Les clics de souris peuvent être extrêmement rapides, on préféra les récupérer dans la boucle d'évènements en même temps que `pygame.QUIT`. Il faudra donc voir si l'un des évènements est l'appui sur le clic gauche :

```
if evenement.type == pygame.MOUSEBUTTONDOWN:
    #
```

Il est alors possible de récupérer la position de la souris avec la fonction `pygame.mouse.get_pos` qui renvoie une paire d'entiers (x, y), la position relative au pixel en haut à gauche :

```
if evenement.type == pygame.MOUSEBUTTONDOWN:
    position = pygame.mouse.get_pos()
    souris_x = position[0]
    souris_y = position[1]
    print(souris_x, souris_y)
```

Évènements aléatoires : le module `random`

Pour introduire des éléments aléatoires dans votre jeu, il vous est possible d'utiliser le module python `random`. Après avoir importé le module avec `import random` dans la partie initialisation, vous pouvez faire appel aux fonctions suivantes :

- `random.randint(a, b)` renvoie un nombre entier uniformément distribué entre `a` et `b` inclus.
- `random.random()` renvoie un nombre à virgule uniformément distribué entre 0 inclus et 1 exclu.

La fonction `random.randint` peut par exemple être utilisée pour simuler le lancer d'un dé à 6 faces :

```
de1 = random.randint(1, 6)
de2 = random.randint(1, 6)
if de1 == 6 and de2 == 6:
    print("Double 6 !")
```

Gestion du temps : le module `time`

Vous pouvez stocker dans une variable l'état actuel de l'horloge grâce à la fonction `time.time`. Vous pouvez ensuite soustraire cette variable à un second appel pour connaître le nombre de secondes écoulées. Par exemple, dans la partie initialisation vous pouvez ajouter :

```
import time
debut = time.time()
```

Puis lors de la gestion des appuis sur les touches du clavier (voir la section 2.7) :

```
if touches[pygame.K_SPACE] == 1:
    print(time.time() - debut, "secondes")
```

Avec ces deux lignes, appuyer sur espace affiche le nombre de secondes écoulées depuis le lancement du jeu.

Fluidité du jeu : fixer la durée d'une boucle

Dans la boucle de jeu, il est parfois nécessaire d'effectuer des opérations complexes qui ralentissent le programme. Par conséquent, le jeu peut paraitre moins fluide, les personnages se déplaçant en saccade. Pour contrer cela, il est possible de donner une durée minimale à un tour de boucle avec un objet horloge de Pygame. Dans la partie initialisation, vous pouvez ajouter la ligne :

```
horloge = pygame.time.Clock()
```

Puis à la fin de la boucle de jeu vous pouvez faire :

```
horloge.tick(60)
```

Où 60 est le nombre de tour de boucle qui seront effectués chaque seconde. Ce nombre doit généralement être plus grand que 30 pour obtenir un jeu un minimum fluide. C'est ce que l'on appelle couramment les *FPS* (*frames per second*, images par seconde). En pratique, la fonction `horloge.tick` va volontairement stopper le programme quelques millisecondes lorsque la boucle tourne trop vite. Après `horloge.tick`, vous pouvez ajouter un `print(horloge.get_fps())` pour vérifier que vous avez bien 60 tours de boucle par seconde.

Saut réaliste : vitesse et gravité

Lorsque vous avez un personnage, sa position sur l'écran peut être vue comme un vecteur $\vec{p}$ en deux dimensions $\vec{p} = (p_x, p_y)$. C'est les deux premières valeurs du rectangle correspondant à la sprites du personnage. On peut également considérer un vecteur $\vec{v}$ qui représente la vitesse de déplacement du personnage. À chaque tour de boucle v_x et v_y sont mis à zéro et si le joueur appuie sur la flèche gauche, v_x est mis à -3 pour que le personnage se déplace de 3 pixels par tour de boucle. Une fois la vitesse calculée, on change le vecteur $\vec{p}$ en $\vec{p} + \vec{v}$ pour déplacer le personnage [3].

3. N'oubliez pas de faire ça en deux fois : d'abord le déplacement horizontalement (p_x est transformé en $p_x + v_x$), suivi de la résolution des collisions (voir section 2.9), puis déplacement vertical, suivi d'une autre phase de résolution des collisions.

Pour avoir un saut réaliste, il ne faut pas remettre v_y à 0 à chaque tour de boucle. Plutôt, il faut l'incrémenter à chaque tour de boucle $v_y \leftarrow v_y + 1$, cela correspond à l'effet de la gravité qui nous pousse en permanence vers le bas. Par ailleurs, lorsque la touche de saut est appuyée, il suffit de donner une impulsion vers le haut au personnage en fixant $v_y \leftarrow -20$. Le saut prendra donc la forme suivante :

- Au premier tour de boucle v_y est -20, le personnage se déplace vers le haut de 20 pixels.
- Au second tour de boucle v_y est affecté par la gravité et passe à -19, le personnage se déplace vers le haut de 19 pixels.
-
- Au vingt-et-unième tour de boucle, la gravité passe v_y de -1 à 0, le personnage ne bouge plus.
- Au vingt-deuxième tour de boucle, la gravité passe v_y de 0 à 1, le personnage commence à retomber de 1 pixel.
- Au vingt-troisième tour de boucle, la gravité passe v_y de 1 à 2, le personnage tombe de 2 pixels.
- Au vingt-quatrième tour de boucle, la gravité passe v_y de 2 à 3, le personnage tombe de 3 pixels.
-

Vous remarquerez que quand le personnage retombe, il gagne de la vitesse à chaque tour de boucle, comme on peut l'observer dans la vie réelle. En pratique le saut va former une parabole, c'est la trajectoire standard d'un objet affecté par la gravité.

Pour gérer ça dans votre code, en plus de stocker la position du personnage comme vous devez déjà le faire, stockez également deux valeurs : la vitesse horizontale et la vitesse verticale. Dans la gestion des évènements, au lieu d'ajouter une valeur directement à la position du personnage, changez la vitesse. Si vous rencontrez une collision en dessous du personnage, n'oubliez pas de mettre la vitesse verticale à 0, cela correspond à toucher le sol.

Mouvements réalistes : Animation des *sprites*

Pour animer un personnage, rien de plus simple : il suffit d'afficher une image différente à chaque tour de boucle du jeu. Regardez les images disponibles en annexe A.4, il y a par exemple 2 images pour un personnage immobile (les images *standing*). Pour les utiliser, il faut commencer par stocker dans une variable le tour de boucle actuel : initialisez une variable `tour_de_boucle` à 0, puis à chaque tour de boucle incrémentez là de 1. Si vous changiez l'image exactement à chaque tour de boucle, l'animation risque d'être beaucoup trop rapide, du coup on préfèrera choisir l'image en fonction de `tour_de_boucle // 50 % 2`, cette valeur sera 0 pour les 50 premiers tours de boucle, puis 1 pour les 50 suivants, puis de nouveau 0 pour les 50 suivants, etc. Du coup, au lieu `blit` toujours la même image du personnage, alternez entre les deux images disponibles selon la valeur de `tour_de_boucle // 50 % 2`.

Pour simplifier l'écriture du code, il est bien plus facile de mettre les images dans un tableau, vous pouvez même utiliser une compréhension de liste (la syntaxe avec un `for` entre des crochets) pour charger toutes les images à la fois [4] :

4. Ici le `%d` sera remplacé par `i` à chaque tour de boucle, c'est une syntaxe particulière pour inclure un nombre dans une chaine de caractères.

```
court = [pygame.image.load(
        "sprites/anime/p1/running/frame-%d.png" % i
    ).convert_alpha()
    for i in range(1,7)]
```

Pour choisir quelle image afficher, il suffit de regarder la vitesse du personnage (voir section 3.6). Par ailleurs, il vous est possible de stocker une variable direction pour savoir dans quelle direction le personnage est orienté. Grâce à la fonction `pygame.transform.flip` (voir section 4.2), il vous est alors possible d'afficher un personnage regardant vers la gauche.

Référence

Touches du clavier

Les lettres sont facilement accessible, par exemple la touche x est désignée par `pygame.K_x`. Il en va de même pour les chiffres et les touches F1 à F12. Pour les autres touches il faut connaitre les traductions anglaises, dans le doute demandez à un organisateur. Les touches directionnelles sont `pygame.K_left`, `pygame.K_right`, `pygame.K_up`, `pygame.K_down`. Les touches espaces et entrée sont respectivement `pygame.K_SPACE` et `pygame.K_RETURN`. Hormis cela, les autres touches disponibles sont :

K_AMPERSAND	K_GREATER	K_KP_PERIOD	K_POWER
K_ASTERISK	K_HASH	K_KP_PLUS	K_PRINT
K_AT	K_HELP	K_LALT	K_QUESTION
K_BACKQUOTE	K_HOME	K_LAST	K_QUOTE
K_BACKSLASH	K_INSERT	K_LCTRL	K_QUOTEDBL
K_BACKSPACE	K_KP0	K_LEFTBRACKET	K_RALT
K_BREAK	K_KP1	K_LEFTPAREN	K_RCTRL
K_CAPSLOCK	K_KP2	K_LESS	K_RIGHTBRACKET
K_CARET	K_KP3	K_LMETA	K_RIGHTPAREN
K_CLEAR	K_KP4	K_LSHIFT	K_RMETA
K_COLON	K_KP5	K_LSUPER	K_RSHIFT
K_COMMA	K_KP6	K_MENU	K_RSUPER
K_DELETE	K_KP7	K_MINUS	K_SCROLLOCK
K_DOLLAR	K_KP8	K_MODE	K_SEMICOLON
K_END	K_KP9	K_NUMLOCK	K_SLASH
K_EQUALS	K_KP_DIVIDE	K_PAGEDOWN	K_SYSREQ
K_ESCAPE	K_KP_ENTER	K_PAGEUP	K_TAB
K_EURO	K_KP_EQUALS	K_PAUSE	K_UNDERSCORE
K_EXCLAIM	K_KP_MINUS	K_PERIOD	K_UNKNOWN
K_FIRST	K_KP_MULTIPLY	K_PLUS	

Les noms de touches préfixés par `K_KP` correspondent aux touches du clavier numérique.

Transformations d'une image

Les fonctions de transformations prennent toute une image en paramètre et vous renvoient une nouvelle image transformée. Il y 3 fonctions qui peuvent vous intéresser.

Redimensionnement avec la fonction `pygame.transform.scale`. Celle-ci prend un second argument : une paire `(w, h)`, la nouvelle taille de l'image.

```
petite_image = pygame.transform.scale(image, (16, 16))
```

Symétries avec la fonction `pygame.transform.flip`. Celle ci prend un total de trois arguments : l'image, est ce qu'elle doit être renversé horizontalement et est ce qu'elle doit être renversé verticalement.

```
regarde_gauche = pygame.transform.flip(personnage, True, False)
```

Rotation avec la fonction `pygame.transform.rotate`. Celle-ci prend un second argument : l'angle de rotation (dans le sens trigonométrique, c'est-à-dire anti-horaire).

```
tombe = pygame.transform.rotate(personnage, 45)
```

Sprites

Terrains

Toutes ces images se trouvent dans `sprites/terrain` et sont de taille 64×64.

Personnages

Toutes ces images se trouvent dans `sprites/personnage` et sont de taille 35×45.


`gnu.png`


`kisi.png`


`kit.png`


`sara.png`


`tux.png`


`wilber.png`

Objets

Toutes ces images se trouvent dans `sprites/objet` et sont de taille 64×64.


`bomb.png`


`coinGold.png`


`flagRed.png`


`star.png`


`switchLeft.png`


`switchMid.png`


`barrel.png`


`bush.png`


`box.png`

Personnages animés

Toutes ces images se trouvent dans `sprites/anime` et sont de taille 50×75.


`p1/standing/frame-1.png`


`p1/standing/frame-2.png`


`p1/running/frame-1.png`


`p1/running/frame-2.png`


`p1/running/frame-3.png`


`p1/running/frame-4.png`


`p1/running/frame-5.png`


`p1/running/frame-6.png`


`p1/jump/jump_up.png`


`p1/jump/jump_fall.png`


`p1/got_hit/frame.png`

Vous trouverez à peu prêt les mêmes sprites pour les deux personnages suivant :

`p2/standing/frame-1.png`	`p3/standing/frame-1.png`
`p2/standing/frame-2.png`	`p3/standing/frame-2.png`
`p2/running/frame-1.png`	`p3/running/frame-1.png`
`p2/running/frame-2.png`	`p3/running/frame-2.png`
`p2/running/frame-3.png`	`p3/running/frame-3.png`
`p2/running/frame-4.png`	`p3/running/frame-4.png`
`p2/running/frame-5.png`	`p3/running/frame-5.png`
`p2/running/frame-6.png`	`p3/running/frame-6.png`
`p2/jump/jump_up.png`	`p3/jump/jump_up.png`
`p2/jump/jump_fall.png`	`p3/jump/jump_fall.png`
	`p3/got_hit/frame.png`