Explications des fichiers du tutoriel

Vincent Pagé.

Introduction

Dans ce document, je vais reprendre les explications que j'ai faites en cours pour bien débuter la programmation d'un jeu vidéo en **python**, utilisant la librairie **pygame**

Attention : 1. Je suppose que les pré-requis ce projet sont remplis. Re-regardez ce lien pour vous en assurer.

2. ce même lien vous donne aussi accès à l'archive des fichiers sur lesquels nous allons travailler. Téléchargez l'archive et décompressez la, vous modifierez les fichiers pour vos tests.

Arrivés au terme de ce document, si vous suivez effectivement mes propositions, vous aurez déja fait une partie du TP. Il serait dommage de vous en priver.

L'archive contient les fichiers suivant : - 01_fenetre.py - 02_escape.py - 03_horloge.py - 04_imagePerso.py - 05_imageFond.py - 06_imageTexte.py - background.jpg - balle.png - perso.png

les trois derniers fichiers sont des images. les 6 premiers sont des programmes écrits en **python**. Ils doivent fonctionner pour **python 2.7** et **python 3**.

A l'université, cette année, vous utiliserez python 2.7. Chez vous, utilisez python 3. Globalement, cela ne changera pas grand chose pour vous.

Les 6 fichiers python sont en fait différentes étapes que j'ai suivies pour obtenir un début de jeu vidéo. A chaque étape, je n'ajoute que quelques lignes.

Voici les 6 étapes : 1. Création d'une fenêtre graphique pour notre jeu : 01_fenetre.py

2. Ajout de la possibilité de quitter le jeu : 02_escape.py
3. Ajout d'une horloge pour cadencer notre jeu : 03_horloge.py
4. Ajout d'une image du personnage dans la fenêtre : 04_imagePerso.py
5. Ajout d'une image de fond dans la fenêtre : 05_imageFond.py
6. Ajout d'un texte fixe dans la fenêtre 06_imageTexte.py

Je vais maintenant prendre chaque fichier pour vous expliquer tout son contenu. A chaque fois, je vous conseille d'ouvrir le fichier dans votre éditeur de texte favori pour suivre ce que je fais.

Attention Il ne s'agit pas d'apprendre ces lignes par coeur, il s'agit de comprendre ce qu'elles font pour les adapter et les modifier plus tard. Les modifications commenceront a partir de l'étape 4.

1. Création de la fenêtre

Attention : si vous exécutez ce programme, il va entrer dans une boucle infinie qu'il faudra quitter en appuyant sur Ctrl+C pour le quitter.

Nous allons essayer de créer une fenêtre qui reste ouverte indéfiniment.

Voyons donc les lignes telles qu'elles arrivent :

```
import pygame
```

Cette ligne dit à python que vous voulez utiliser la bibliothèque **pygame** (on parle du **module pygame** en python).

Si pygame n'est pas installé sur votre machine le programme s'arrêtera là avec une erreur.

```
# Initialisation de la bibliotheque pygame
pygame.init()
```

Le # est un commentaire en python. Cette ligne est ignorée par le programme. La phrase *Initialisation ....* est juste là pour aider la personne qui regarde mon fichier python à comprendre ce que je veux faire.

pygame.init() est un appel de la fonction *init()* de pygame qui va se charger de préparer le terrain pour notre jeu. *(si cela vous intéresse, sachez que pygame va regarder quelles sont les propriétés de votre écran, carte graphique, carte son ... parce que notre programme va jouer avec. Mais nous ne nous en rendrons pas compte)*

```
# creation de la fenetre
largeur = 640
hauteur = 480
fenetre=pygame.display.set_mode((largeur,hauteur))
```

Ces lignes vont créer une fenêtre de 480 pixels de hauteur par 640 pixels de largeur.

les variables *largeur* et *hauteur* ne doivent pas vous poser de problèmes (sinon retournez voir le cours de python)

Cette ligne mérite quelques explications supplémentaires

```
fenetre=pygame.display.set_mode((largeur,hauteur))
```

la fonction *set_mode()* va créer une fenêtre. J'aurais plus loin besoin de cette fenêtre (pour dessiner dedans). Je récupère donc la réponse de la fonction pour la mettre dans une variable que j'ai appelé *fenetre*.

Vous pouvez noter que *fenetre* n'est sans doute pas un type simple comme les entiers, float, string que vous avez sans doute vu avant. C'est vrai. c'est un **objet** (complexe). Mais on s'en moque. De notre point de vue, c'est juste la variable qui représente notre fenêtre.

A ce stade, si mon programme ne contenait que cela, je pourrais le lancer, les lignes s'exécuteraient jusque là, la fenêtre s'ouvrirait. Puis mon programme se termine, et il ferme la fenêtre avec lui...

Nous avons crée le jeu vidéo le plus court du monde (ou presque)

Pour que le jeu reste ouvert, il faut ajouter une boucle. qui s'exécute jusqu'à ce que l'utilisateur en ait assez.

Au plus simple, je vais ajouter une boucle infinie comme suit.

```
# la boucle infinie dans laquelle on reste coince
i=1;
continuer=1
while continuer=1:
    i= i+1;
    print (i)
```

Dans les lignes qui précèdent, vous avez une boucle classique qui s'exécute tant que *continuer* a la valeur *1*. Si vous ne comprenez pas ce code, retournez voir la partie cours ICI.

Évidemment, comme mon code est entré dans une boucle infinie, je n'atteindrais jamais la suite du code. Je vous l'explique quand même car elle servira plus tard

```
# fin du programme principal.
# On n'y accedera jamais dans le cas de ce programme
pygame.quit()
```

Normalement, quand un programme utilisant pygame se termine, il appelle la fonction *quit()* avant de se terminer (cela range bien tout comme il faut).

Vous trouverez ici le fichier complet de cette étape.

2. Sortir du jeu

Je n'ai ajouté que quelques lignes de code dans le programme précédent. Essayez de les repérer avant de passer à la suite, je vais les analyser avec vous.

L'objectif, je vous le rappelle, est de pouvoir sortir de notre programme quand l'utilisateur appuie sur la touche *Escape* de son clavier ou quand il clique sur le bouton *fermer* de la fenêtre de notre programme.

Ceci ce produit alors que notre fenêtre est déja ouverte, donc au coeur de notre boucle *while*

Traitement des touches enfoncées.

le premier bloc de nouveautés est le suivant et concerne la détection d'un appui sur la touche *Escape*.

```
# on recupere l'etat du clavier
touches = pygame.key.get_pressed();
```

key.get_pressed() est une fonction de pygame. Son nom m'indique plus où moins à quoi elle sert : Elle va me répondre quelles touches du clavier sont enfoncées au moment où je l'appelle. Ma ligne met le résultat dans une variable que j'ai nommé *touches*

Pour traiter le cas qui m'intéresse (quand la touche *Escape* est enfoncée), il faut connaître la tête de ma variable *touches*. La doc de pygame peut nous renseigner là dessus et en voici une explication très brève.

La réponse de la fonction *key.get_pressed()* est un tableau de booléens (des variables vraies ou fausse, True ou False en python). Chaque case du tableau est une touche du clavier. La valeur d'une case est *True* si la touche est enfoncée et *False* si la touche n'est pas enfoncée au moment de l'appel.

Le tableau peut être représenté comme suit

indice	valeur
0	False
1	False
...	False
42	True
...	False

indice	valeur
63	True
...	False

Ici, on voit que les touches numéro 42 et 63 étaient les seules enfoncées au moment ou l'appel de la fonciton a été fait.

Il ne me reste plus qu'a regarder les touches qui m'intéressent. Si je sais que la touche *ESC* porte le numéro 43, je pourrais faire quelque chose comme :

```
if touches[43] == True:
  print "Vous avez appuyé sur Escape"
```

Mais les développeurs ne veulent pas apprendre par coeur le numéro des touches. Pygame leur fournit donc une assistance. Le numéro de la touche *Esc* est par exemple contenu dans une constante nommée *K_ESC*. Le code précédent deviendrait donc :

```
if touches[pygame.K_ESC] == True:
  print "Vous avez appuyé sur Escape"
```

Voici une petite liste de ces constantes :

- K_ESCAPE : la touche escape.
- K_RETURN : la touche d'entrée.
- K_SPACE : vous savez
- K_A : la touche A
- K_LEFT : la touche "flêche de gauche"

Adaptons ceci a notre cas... Ce qui nous intéresse est, lorsque la touche Escape est enfoncée, de quitter la boucle.

Plus précisément, quand la touche Escape est enfoncée, **nous n'allons pas instantanément quitter la boucle**. Nous arrêterons la boucle au début du prochain tour (on parle de **la prochaine itération**).

Pour cela, il suffit de rendre faux le test du *while*. Nous mettrons donc *continuer* à une valeur différente de *0*.

Ceci explique le second bloc de nouveauté dans le fichier, présenté ci-dessous

```
# si la touche ESC est enfoncee, on sortira
# au debut du prochain tour de boucle
if touches[pygame.K_ESCAPE] :
    continuer=0
```

Remarque : juste ces modifications ne donneront pas un programme fonctionnel pour des raisons que je ne détaillerais pas ici. Il faudra ajouter la suite pour que l'ensemble fonctionne. Si cela vous intéresse, demandez moi, mais vous pouvez vivre sans le savoir.

Occupons nous maintenant du bouton de fermeture de l'application.

Traitement du bouton QUIT

Il me faut ici vous parler du fonctionnement général de l'ordinateur lorsque l'on utilise des interfaces graphiques.

Nous attendons ici que notre programme réagisse à une action. Néanmoins, nous sommes ici très loin de l'attente de l'entrée d'un entier au clavier, car :

- nous ne savons pas quand cette action va se produire.
- notre programme ne doit pas être bloqué pendant l'attente.

Quand vous déplacez la souris, cliquez quelque part (ou même appuyez sur une touche du clavier), la seule entité qui soit au courant de cette action est votre système d'exploitation (votre **OS**, qui peut être Windows, Mac OS ou un Linux ou . . .)

Prenons l'exemple d'un clic souris :

Votre **OS** sait quelle est l'application qui se trouve sous l'événement (il gère le positionnement de vos fenêtres sur votre écran).

Il va donc envoyer un message à l'application concernée. Ce message, correspondant à une action faite par l'utilisateur est appelé un **événement**.

Un programme qui utilise une interface graphique doit donc gérer ces événements.

un événement a plusieurs propriétés : - un **type**. Voici quelques exemples :

- Mouse_down : le bouton de gauche de la souris a été enfoncé.
- Mouse_up : le bouton de gauche de la souris a été enfoncé.
- Mouse_move : la souris a bougé
- Key_down : une touche a été enfoncée
- des valeurs utiles :
  - pour un Mouse_up, il nous faut sa position.
  - Pour un Key_down, il nous faut savoir quelle touche est concernée

Dans notre cas, le **type** de l'événement qui nous intéresse est *QUIT* qui correspond au clic sur le bouton de fermeture. Cet événement n'a pas de valeur importante pour nous.

pygame dispose d'une fonction (*event.get*) qui renvoie un tableau de tous les événements qui se sont produits depuis la dernière fois qu'on l'a demandé.

Voici ce que nous allons faire. A la fin de chaque tour de boucle :

1. on récupère la liste des événements.
2. on parcours cette liste à la recherche d'un événement de type QUIT.
3. Si on tombe sur un événement de type QUIT, on met *continuer* à *0* ce qui arrêtera la boucle au début de la prochaine itération.

Voici le code qui fait cela :

```
# Si on a clique sur le bouton de fermeture on sortira
# au debut du prochain tour de boucle
# Pour cela, on parcours la liste des evenements
# et on cherche un QUIT...
```

```
for event in pygame.event.get(): # parcours de la liste des evenements recus
    if event.type == pygame.QUIT: # Si un de ces evenements est de type QUIT
        continuer = 0
```

Vous trouverez ici le fichier complet de cette étape.

3. Ajout d'une horloge

Il est temps d'affiner un peu ce qu'est notre boucle *while*, à part un moyen d'éviter que la fenêtre se referme...

Pensez à notre jeu comme à un jeu par tours : dans un tour (qui prendra un certain temps), il faudra au final :

- regarder s'il faut déplacer le personnage joueur
- déplacer les ennemis du joueur
- ré-afficher l'écran
- regarder si le joeur veut quitter le programme.

et on recommence...

Ceci est pris en charge par notre boucle *while*. C'est pour cette raison qu'a chaque tour de boucle, on *regarde si le joueur veut quitter le programme*, ce que nous avons fait à l'étape précédente.

Nous allons, au cours du projet, ajouter toutes les étapes listées ci-dessus.

Le problème est que sans contrôle supplémentaire, les tours de boucle se font aussi vite que notre ordinateur peut le supporter.

Sur un ordinateur rapide, les ennemis vont donc se déplacer à toute vitesse. Sur un ordinateur lent, les ennemis auront tendance à lambiner.

Pour éviter cela, nous allons ajouter une horloge qui va servir à **définir la durée minimale** d'un tour de boucle.

Pygame utilise la notion de *Clock* (horloge). Je vais donc, au début de mon programme, demander la création d'une variable de type *Clock*. Je vais nommer cette variable *horloge* :

```
# servira a regler l'horloge du jeu
horloge = pygame.time.Clock()
```

Cette variable est encore une fois d'un type plus étrange qu'un entier (comme *fenetre*, c'est un objet. On dirait ici que *horloge* est une instance de la classe *Clock*)

Il se trouve que les objets ont des fonctions internes. Les objets de type Clock disposent d'une fonction nommée *tick()* qui fonctionne comme suit :

- quand on l'appelle pour la première fois : elle note l'heure (à la milliseconde près)
- quand on la rappelle :
  - elle note l'heure actuelle, et la compare à l'heure précédemment notée.
  - si la différence entre les heures est supérieure à la durée minimale souhaitée, elle ne fait rien.
  - sinon, elle endort le programme jusqu'à ce que cette durée soit atteinte.

Notez que la durée minimale est communiquée a la fonction *tick* en la passant entre parenthèses. On dit qu'on a **passé un argument à la fonction**.

Appliquons cela à notre cas : Nous voulons limiter la vitesse de notre boucle, nous allons donc placer l'appel à la fonction *tick* de notre *horloge* dans la boucle *while* (peu importe où).

Pour définir la durée on passe à la fonction l'inverse de la durée souhaitée : - pour une durée de 0.5s, on passe 2 à la fonction *tick* - pour une durée de 0.1s, on passe 10 à la fonction *tick* - ...

Notre code à insérer pour faire au maximum 2 tours par seconde est donc :

```
# fixons le nombre max de frames / secondes
horloge.tick(2)
```

Dans le jeu final, vous utiliserez sans doute une valeur de 30 tours par seconde.

```
# fixons le nombre max de frames / secondes
horloge.tick(30)
```

Vous trouverez ici le fichier complet de cette étape.

4. Ajout d'une image du personnage

Cette étape est la dernière vraiment importante de ce tutoriel. Vous pourrez presque faire tout le projet avec. Notamment, l'étape suivante peut en être déduite.

A la fin de cette étape, je vous demanderais de faire des choses tout seul pour tester.

Il s'agit ici d'afficher une image dans la fenêtre. Voyons cela plus précisément :

- nous avons sur le disque dur un fichier "perso.png" contenant l'image.
- notre programme a une fenêtre graphique (pour notre programme cette fenêtre est stockée dans une variable nommée *fenetre*)
- on voudrait dessiner le contenu du fichier dans la fenêtre, a une position que notre programme va contrôler.

Programmer, c'est manipuler des variables, souvent par l'intermédiaire de fonctions.

Lecture de l'image.

La première chose à faire, c'est de lire le fichier "perso.png", extérieur au programme, et à en stocker le contenu dans une variable (ici encore, c'est une variable de type *objet*) dont on se servira plus tard.

Cette lecture peut se faire une seule fois au début du programme. Il ne serait pas très malin de le faire a chaque tour de boucle...

Voici le code correspondant.

```
# lecture de l'image du perso
imagePerso = pygame.image.load("perso.png").convert_alpha()
```

mon image est maintenant stockée dans la variable nommée *imagePerso*.

Variable de positionnement

Il faut comprendre que la mémorisation de la position et l'action d'affichage sur l'écran sont deux choses différentes. Concentrons nous sur la première :

Pour stocker la position de notre personnage, nous avons 2 variables à stocker : un x et un y, en pixels, correspondant à la position souhaitée de notre image dans la fenêtre.

Mais notre image est un rectangle. Si je choisis une position $x = 10$, $y = 20$, est ce la position du centre de l'image ou du coin supérieur gauche ou d'un autre point ?

Dans pygame (et la plupart des librairies du même type), on considère toujours la position **du coin supérieur gauche**.

De plus, on va stocker ces informations (x et y) au sein d'une variable plus compliquée : un objet de type *Rect*, qui contiendra x et y mais aussi la largeur et la hauteur de mon rectangle.

pour un rectangle r existant, 4 données nous intéressent : - $r.x$: l'abscisse du point supérieur gauche. - $r.y$: l'ordonnée du point supérieur gauche. - $r.w$: la largeur du rectangle - $r.h$: la hauteur du rectangle

Ces 4 données sont des entiers. si je veux afficher la largeur d'un rectangle nommé r, je taperais :

```
print (r.x)
```

si je veux donner à mon rectangle la position verticale 37, je ferais comme suit :

```
r.y = 37
```

Reste à obtenir un rectangle... Le plus simple est de demander à l'image que l'on veut positionner de nous le fournir.

Notre variable *imagePerso* est un objet capable de fournir un rectangle avec hauteur et largeur pré-remplies.

```
# creation d'un rectangle pour positioner l'image du personnage
rectPerso = imagePerso.get_rect()
```

Nous disposons donc d'une variable de type *Rect* nommée *rectPerso* pour **stocker** position voulue du personnage.

Disons que nous voudrons dessiner plus loin notre personnage en $x = 60$ et $y = 80$, cela se fait donc comme ceci :

```
rectPerso.x = 60
rectPerso.y = 80
```

Je fais ceci avant le début de ma boucle, ce qui définit la position du personnage au début du jeu.

Affichage dans la fenêtre

J'ai une variable contenant l'image (*imagePerso*), une variable contenant la fenêtre (*fenetre*) et une variable contenant la position souhaitée (*rectPerso*).

Voici la ligne qui demande l'affichage :

```
# Affichage Perso
fenetre.blit(imagePerso, rectPerso)
```

En termes informatiques, j'appelle la fonction *blit* de l'objet *fenetre* qui se charge de dessiner.

Cette fonction de dessin prend comme paramètres l'image et le rectangle de positionnnement.

Enfin, il reste une ligne mystérieuse qui ne sera faite qu'une fois après le ou les *blit*

```
# rafraichissement
pygame.display.flip()
```

Imaginons que ma fonction blit dessine directement sur l'écran. Dans un jeu vidéo contenant 50 objets, je vais dessiner 50 fois sur l'écran. Cela poserait des problèmes.

Pour être plus efficace, la fonction *blit* ne dessine pas vraiment sur l'écran. Elle dessine dans une version cachée de la fenêtre.

Je peux donc dessiner 50 fois sur la version cachée de la fenêtre sans toucher l'écran. Une fois tous mes dessins finis, je révèle ma version cachée.

C'est ce que fait la fonction *display.flip*

Important : Ces étapes sont fait dans la boucle *while* donc a chaque tour de jeu, je redessine le personnage. Comme sa position n'a pas bougé, je le redessine au même endroit.

Vous trouverez ici le fichier complet de cette étape.

Jouons un peu

Vous en savez suffisamment pour faire beaucoup de choses...

Un personnage qui bouge tout seul

Vous savez déja que la position horizontale du personnage est stockée dans la variable *rectPerso.x*

Si j'ajoute cette ligne quelque part dans la boucle while :

```
rectPerso.x = rectPerso.x + 5
```

A chaque tour de boucle, j'augmente la position horizontale de mon personnage de 5.

la partie qui *blite* n'a pas changé. Simplement, je dessine mon personnage a une position différente à chaque fois. Mon personnage bouge tout seul...

Si vous n'aimez pas les "traces" que laisse le personnage, cela sera réglé à l'étape suivante. Vous pouvez la consulter tout de suite ou essayer d'autres choses...

Si vous n'aimez pas voir votre personnage partir vers l'infini (et au-delà), vous réglerez cela au cours du projet...

Que se passe-t-il si vous remplacez ce code par :

```
rectPerso.x = rectPerso.x - 5
```

Un personnage qui bouge avec le clavier.

Nous avons vu que l'on pouvait savoir si la touche *Escape* est enfoncée dans l'étape 2.

Nous allons nous servir de cela pour déplacer le personnage seulement quand l'utilisateur appuie sur la fleche de droite. Vous devriez déja savoir le faire. Essayez.

Correction :

Il suffit en fait de mettre ce code dans le *while*

```
if (touches[pygame.K_RIGHT])
  rectPerso.x = rectPerso.x + 5
```

Ajouter une autre image sur l'écran

Disons que l'on veuille ajouter une image de balle (fichier "balle.png") à la position *x=200*, *y=200*

Il suffit de refaire ce que nous avons fait pour le personnage avec les 3 étapes :

1. lecture de l'image de la balle (au dessus du while)
2. création d'un rectangle pour la balle (au dessus du while) et positionnement de ce rectangle.
3. *blit* de l'image dans la fenêtre (dans le while, avant le *display.flip*)

Je vous laisse tester tout cela.

Ajouter un autre Personnage sur l'écran

Imaginons que nous voulions placer un second personnage dans la fenêtre. C'est le même problème que ci dessus.

Mais nous avons déja réalisé l'étape 1. Nous n'avons pas besoin de relire l'image. il nous faut simplement faire :

1. la création du rectangle pour le second personnage et positionnement de ce rectangle.
2. *blit* dans la fenêtre

5. Ajout d'une image de fond

Nous voulons ajouter une image de fond à notre fenêtre.

Cette partie est très simple, car il s'agit simplement d'ajouter une seconde image dans la fenêtre. Nous l'avons déja fait.

```
# lecture de l'image du fond
imageFond = pygame.image.load("background.jpg").convert()

# creation d'un rectangle pour positioner l'image du fond
rectFond = imageFond.get_rect()
rectFond.x = 0
rectFond.y = 0
```

et dans la boucle *while*

```
# Affichage du fond
fenetre.blit(imageFond, rectFond)
```

important : ce dernier bloc doit être positionné avant le *blit* du personnage... (pourquoi ?)

Et voila !

Vous trouverez ici le fichier complet de cette étape.

Remarques

Notez ce qui se passe si vous ajoutez maintenant le déplacement du personnage (automatique ou au clavier) que je proposais dans l'étape précédente.

A chaque étape, on : 1. redessine tout le fond 2. redessine le Personnage

De ce fait, les "traces" du personnage sont écrasées par le nouveau dessin du fond.

Si vous trouvez inutile de redessiner tout le fond, alors que seuls quelques pixels du fond doivent être re-dessinés, sachez que c'est légitime (mais un poil compliqué à mettre en oeuvre).

je vous inviterais dans ce cas à lire la partie de la documentation de pygame qui parle de **dirty rect animation**. On la trouve ici.

6. Ajout de texte dans la fenêtre

Nous allons maintenant écrire du texte dans la fenêtre. Cela pourrait vous servir pour afficher le score pendant le jeu (ou tout autre message).

Dans la fenêtre, pygame sait dessiner des images (on les appelle **surfaces** selon la terminologie de pygame mais on s'en moque)

Il s'agit donc de créer une image contenant le texte.

L'image que l'on veut créer dépend de la police choisie qui prend en compte trois choses : - la famille de police utilisée - la taille du texte à afficher - la couleur

Encore une fois, votre programme manipule des variables... Créons donc une police adaptée avant le début de la boucle *while*.

```
## Ajoutons un texte fixe dans la fenetre :
# Choix de la police pour le texte
font = pygame.font.Font(None, 34)
```

ma police, stockée dans la variable *font* crée par le code précédent va me servir à générer une image de texte :

```
# Creation de l'image correspondant au texte
imageText = font.render('<Escape> pour quitter', True, (255, 255, 255))
```

Dans le code qui précède, l'objet *font* propose une fonction *render* qui crée l'image. Cette fonction a besoin de 3 paramètres :

- *' < Escape > pour quitter'* est le texte à afficher.
- *True* est un truc sans intérêt (un booléen pour choisir ou non l'option d'anti-aliasing)
- *(255, 255, 255)* est un triplet de valeur qui définit une couleur RGB. Ici, c'est du blanc. *(255,0,0)* serait du rouge

Le reste a déja été vu :

- on crée un rectangle, qu'on positionne.

```
# creation d'un rectangle pour positioner l'image du texte
rectText = imageText.get_rect()
rectText.x = 10
rectText.y = 10
```

- on *blit* l'image du texte.

```
# Affichage du Texte
fenetre.blit(imageText, rectText)
```

Vous trouverez ici le fichier complet de cette étape.

Conclusion et Remarques

Nous voici au terme de ce tutoriel.

Je ne peux que vous recommander d'essayer de jouer un maximum avec ce code **avant** le début des TP pour avoir une bonne note.

Je ne peux que vous recommander d'essayer de jouer un maximum avec ce code **après** la fin des TP pour vous améliorer.

- C'est en codant qu'on apprend à coder.
- C'est en faisant des choses que l'on trouve rigolotes qu'on trouve la force d'apprendre des choses difficiles.

J'espère que ce projet vous plaira et vous motivera.

Au cours de ces pages, j'ai parfois passé sous silence tout un tas de notions importantes. Cela pourrait potentiellement faire hurler les puristes.

Mon seul objectif était de vous donner les bases pour pouvoir travailler. Nous reviendrons sur ces notions lorsque vous aurez déjà suffisamment pratiqué pour en comprendre les tenants et les aboutissants.

Vous pouvez revenir au Sommaire du projet
