Python avancé

Semaine 1

Objectifs

- réaliser un **vrai** programme Python
- réfléchir à un problème **concret**
- **organiser** son code
- faire un premier rendu sous `git`

Prérequis

- avoir `git` installé
- et avoir retenu quelques bases des cours `git`
- avoir `python` et `conda` installé
- et avoir retenu quelques bases du *primer*

et ça va très bien se passer, car n'oublies pas...

Harry – yer a wizard.

★

Hagrid

I'm a what?

★

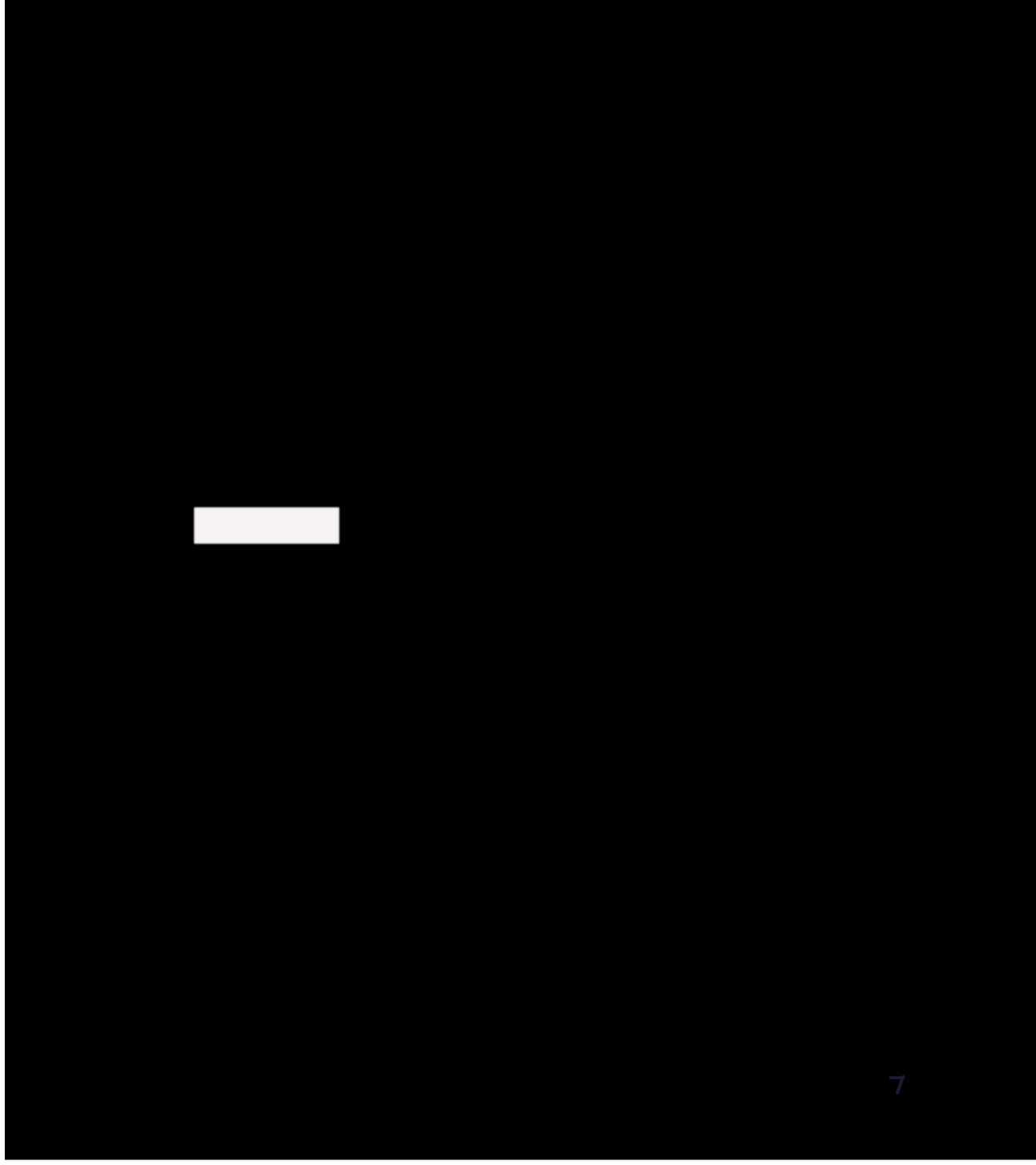
Harry

Partie 1: LE PROJET

Le jeu du serpent

Simplifié

- le serpent est une *suite de blocs*
- à chaque fois qu'il mange un oeuf, il gagne un nouveau bloc
- si il se mord la queue, c'est perdu




Setup

on utilise `conda`[1] et `pip` comme pour le primer:

```
$ conda create -n pas1 python=3.7
$ conda install pip
$ pip install pygame
```

[1] cf. Managing Environments dans la doc conda

on utilise conda[1] et pip comme pour le primer:

```
$ conda create -n pas1 python=3.7
$ conda install pip
$ pip install pygame
```

[1] cf. Managing Environments dans la doc conda

et on vérifie que l'installation fonctionne

```
$ python -m pygame.examples.aliens
```

expliquer que l'on appelle un package "exécutable" (on verra plus tard le __name__ == "**main**" et compagnie)

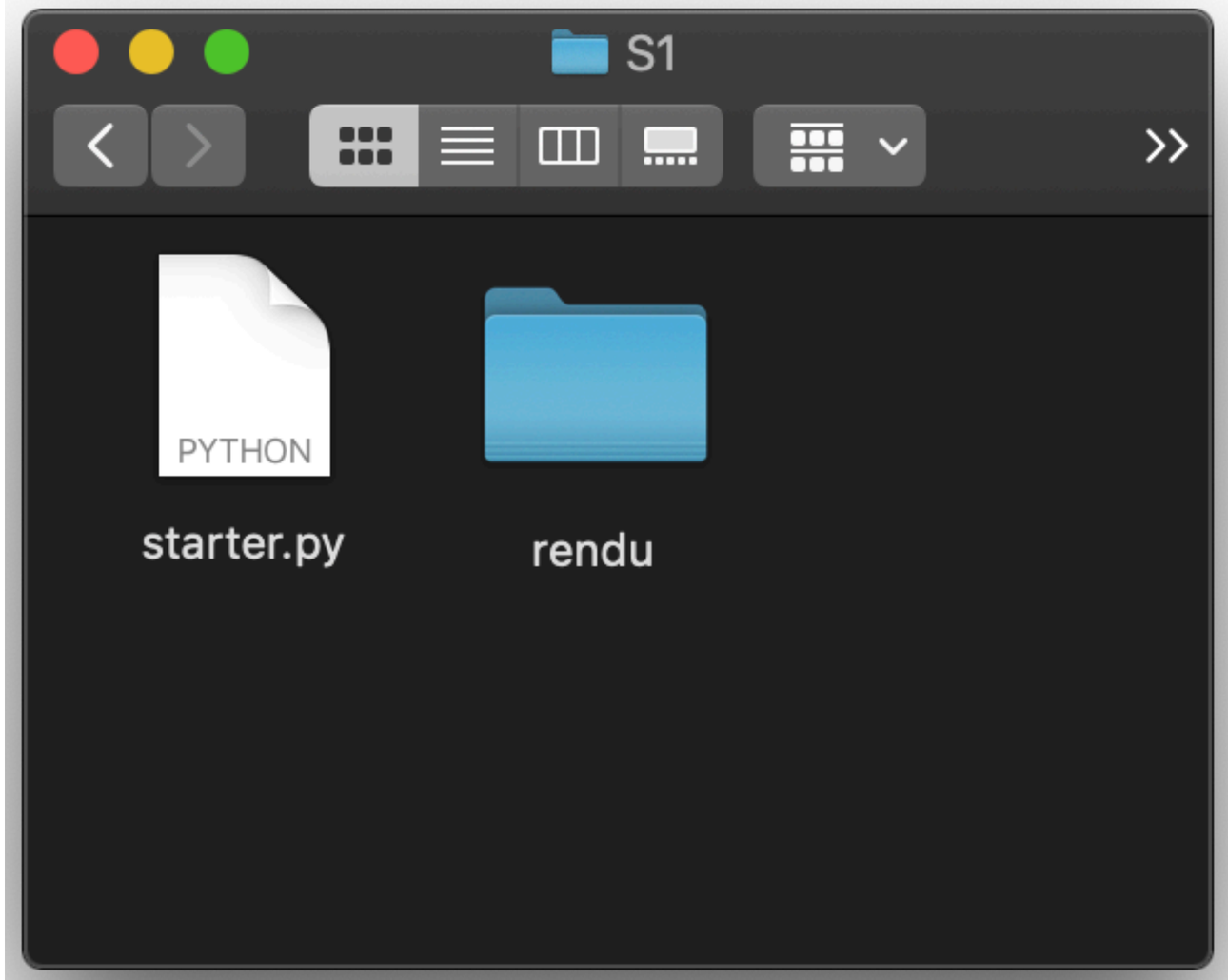
enfin on récupère le projet "starter"

https://pythongroupe3/pythonadvanced

1. on **fork** le projet sur compte Github *personnel*
2. on **clone** le projet localement
3. on **active** l'environnement conda pas1

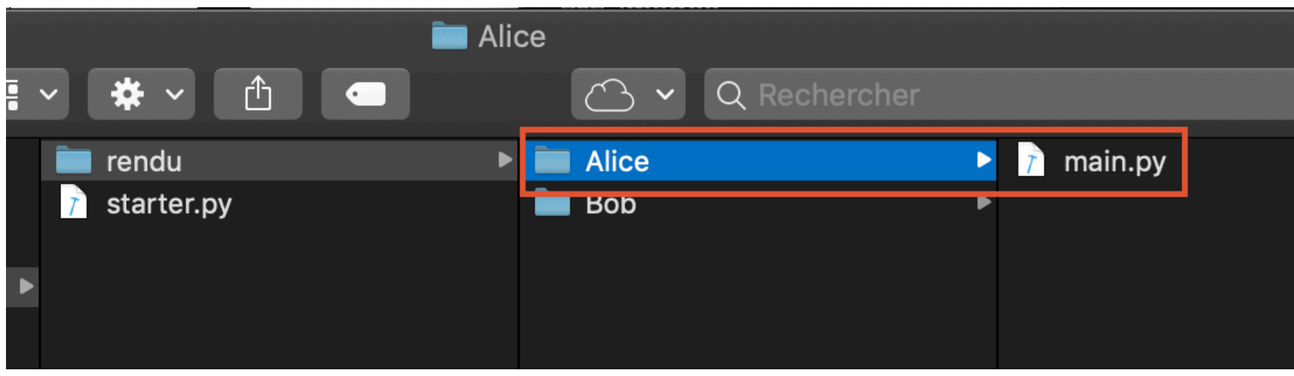
contenu du dépôt git

- un fichier de démarrage `starter.py`
- un dosser `rendu`



ce qui est attendu

- créer un sous-répertoire de `rendu` à votre prénom
- travailler dedans et faire des `commits` après **chaque exercice**
- à la faire du cours, `push`er votre travail vers GitHub et faire une **Pull Request**




PENSEZ À COMMITER

```python
import sys
import pygame
from pygame.locals import *

# on doit "initialiser" PyGame
pygame.init()
# et définir la taille de la fenêtre (400x400)
screen = pygame.display.set_mode((400, 400))

while True:
  for event in pygame.event.get(KEYDOWN):
    if event.key == K_q: # pygame.locals.K_q
      sys.exit() # quitte le programme
```

```python
import sys
import pygame
from pygame.locals import *

# on doit "initialiser" PyGame
pygame.init()
# et définir la taille de la fenêtre (400x400)
screen = pygame.display.set_mode((400, 400))

while True:
  for event in pygame.event.get(KEYDOWN):
    if event.key == K_q: # pygame.locals.K_q
      sys.exit() # quitte le programme
```

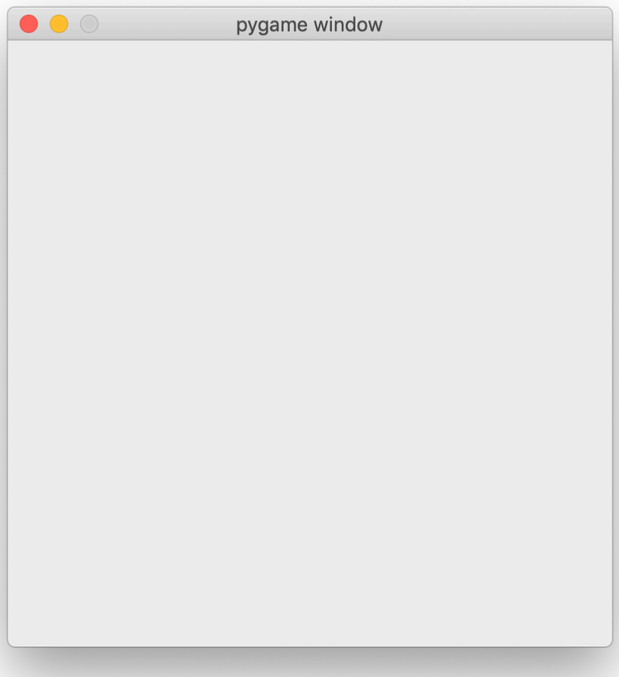
```python
import sys
import pygame
from pygame.locals import *

# on doit "initialiser" PyGame
pygame.init()
# et définir la taille de la fenêtre (400x400)
screen = pygame.display.set_mode((400, 400))

while True:
    for event in pygame.event.get(KEYDOWN):
        if event.key == K_q: # pygame.locals.K_q
            sys.exit() # quitte le programme
```

```python
import sys
import pygame
from pygame.locals import *

# on doit "initialiser" PyGame
pygame.init()
# et définir la taille de la fenêtre (400x400)
screen = pygame.display.set_mode((400, 400))

while True:
  for event in pygame.event.get(KEYDOWN):
    if event.key == K_q: # pygame.locals.K_q
      sys.exit() # quitte le programme
```

```python
import sys
import pygame
from pygame.locals import *

# on doit "initialiser" PyGame
pygame.init()
# et définir la taille de la fenêtre (400x400)
screen = pygame.display.set_mode((400, 400))

while True:
  for event in pygame.event.get(KEYDOWN):
    if event.key == K_q: # pygame.locals.K_q
      sys.exit() # quitte le programme
```

```python
import sys
import pygame
from pygame.locals import *

# on doit "initialiser" PyGame
pygame.init()
# et définir la taille de la fenêtre (400x400)
screen = pygame.display.set_mode((400, 400))

while True:
  for event in pygame.event.get(KEYDOWN):
    if event.key == K_q: # pygame.locals.K_q
      sys.exit() # quitte le programme
```

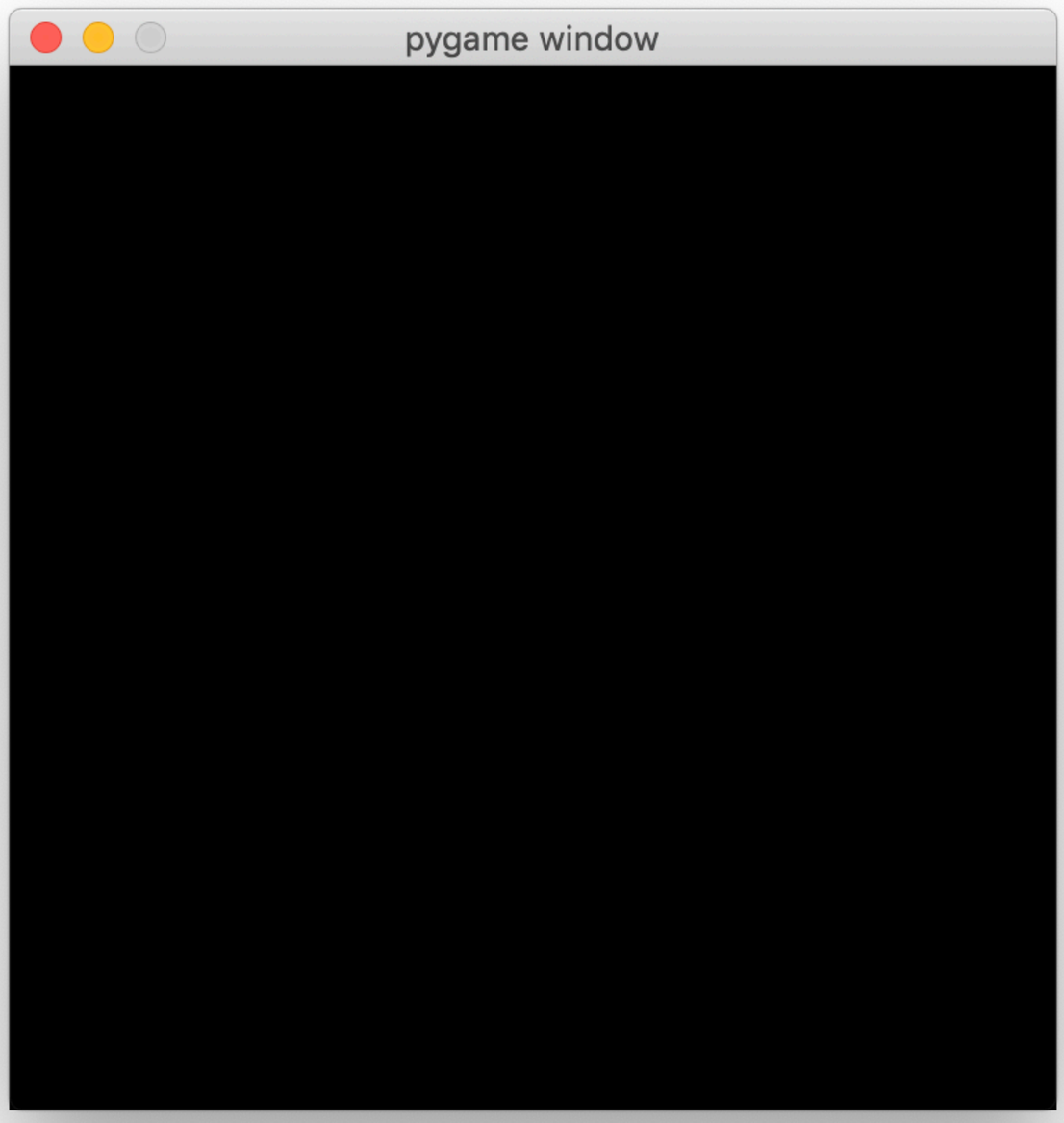
```
$ python main.py
```



Dessinons

Un fond noir[2]

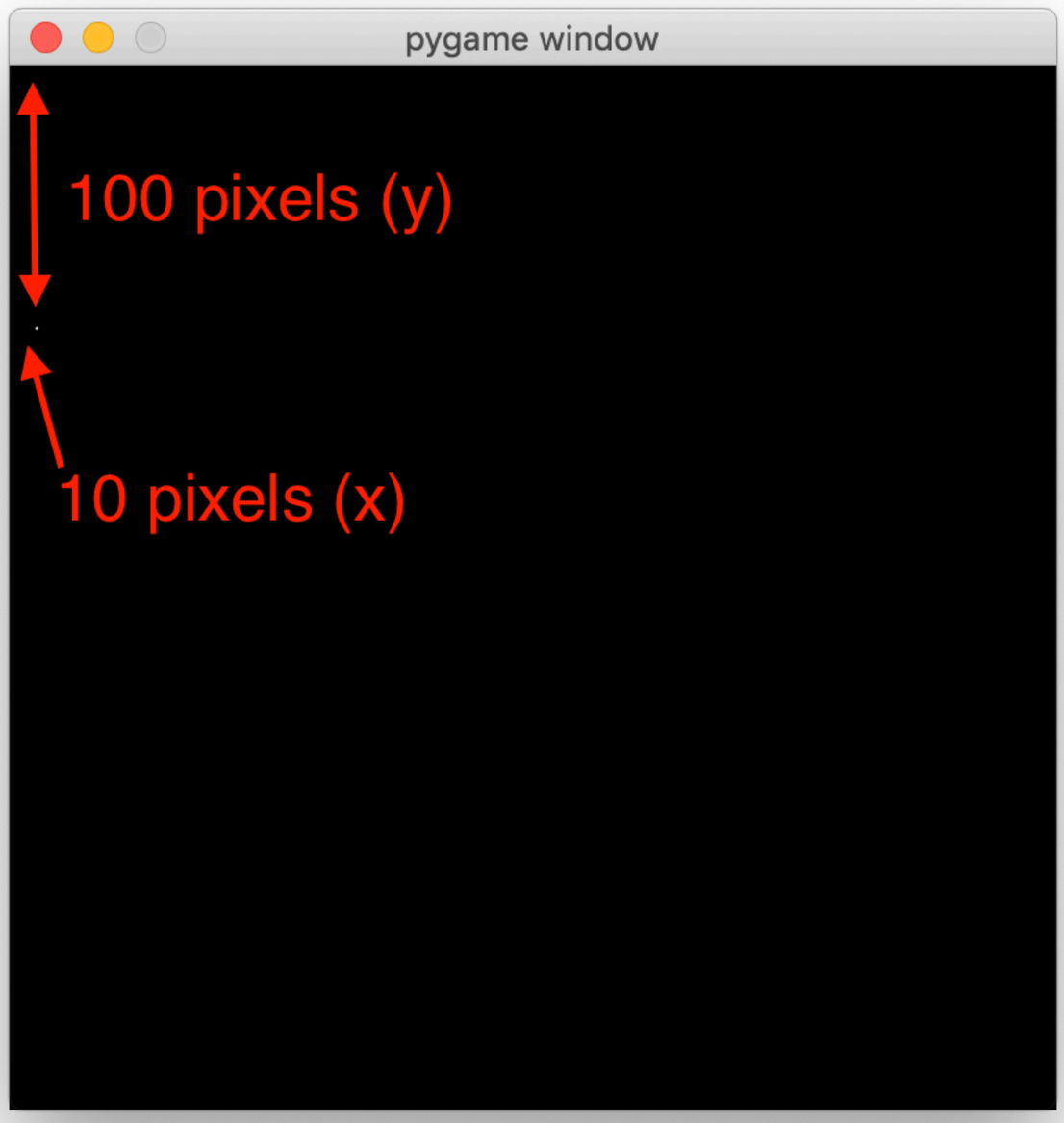
```
black = (0, 0, 0)
screen.fill(black)
pygame.display.update()
```



[2] les couleurs sont en `(r, g, b)` (rouge – vert – noir)

Et un pixel blanc

```
white = (255, 255, 255)
screen.set_at((10, 100), white)
```



Exercice: la cellule

- on découpe l'écran en 40 lignes & 40 colonnes
- chaque cellule fait donc 20 pixels de large et de haut

Écrire une fonction `draw_cell(pos, color)` qui recoit une position `(x,y)`[4] et remplit le carré de 20x20 correspondant avec `color`.

[4] en coordonnées "plateau" $0 \leq x, y < 20$

```python
CELL_SIZE = (20, 20)

def draw_cell(board_pos, color=(255, 255, 255)):
  x, y = board_pos
  cell_width, cell_height = CELL_SIZE
  for line in product(range(cell_height)):
    for col in product(range(cell_width)):
      screen_pos = (cell_width * x + col, cell_height * y + line)
      screen.set_at(screen_pos, color)
```


je ferai un dessin...

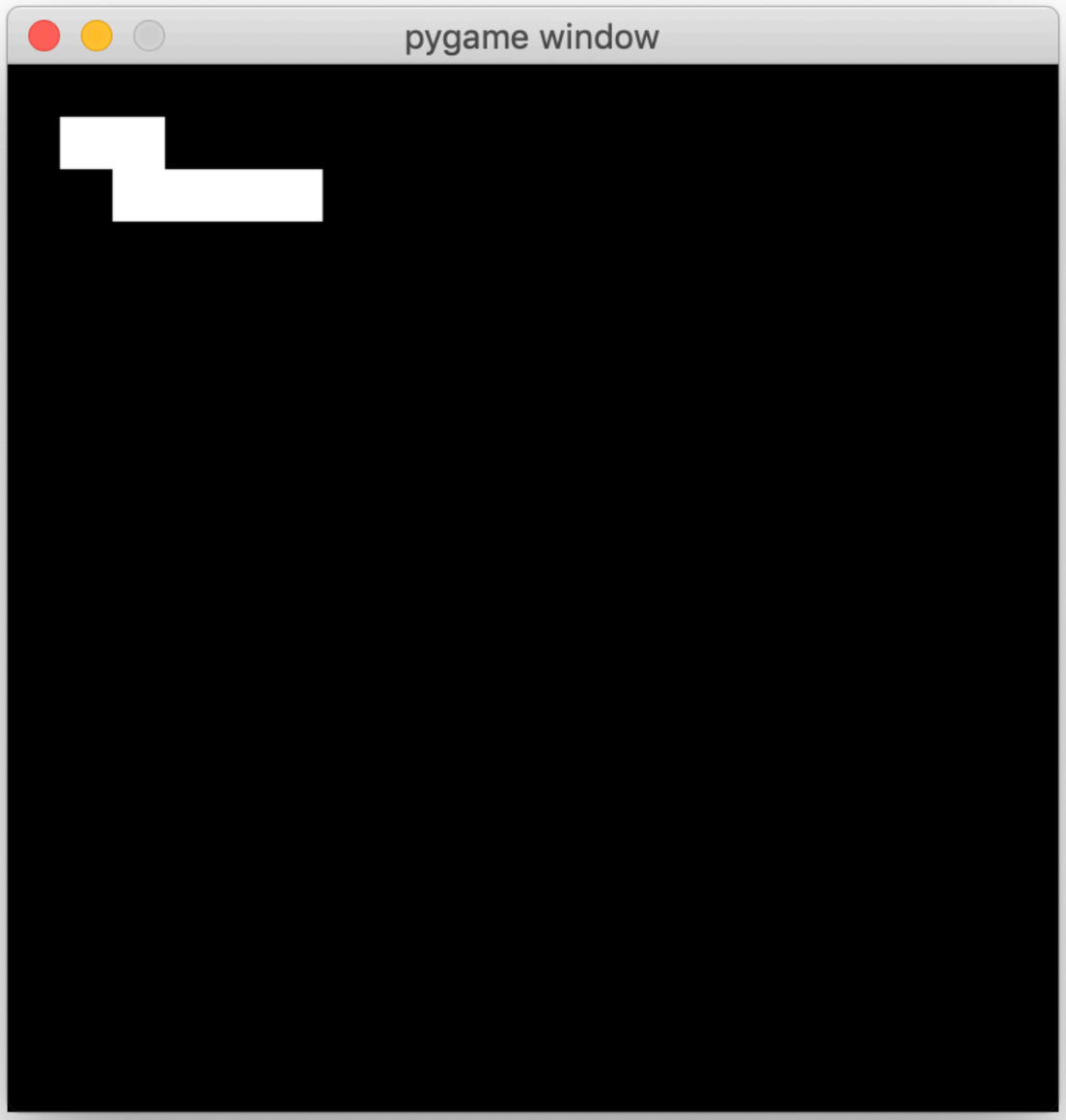

COMMIT

Exercice: le serpent

on définit le serpent comme une suite de positions:

```
snake = [
  (1, 1),
  (2, 1),
  (2, 2),
  (3, 2),
  (4, 2),
  (5, 2)
]
```

Tracer la position actuelle du serpent à l'écran

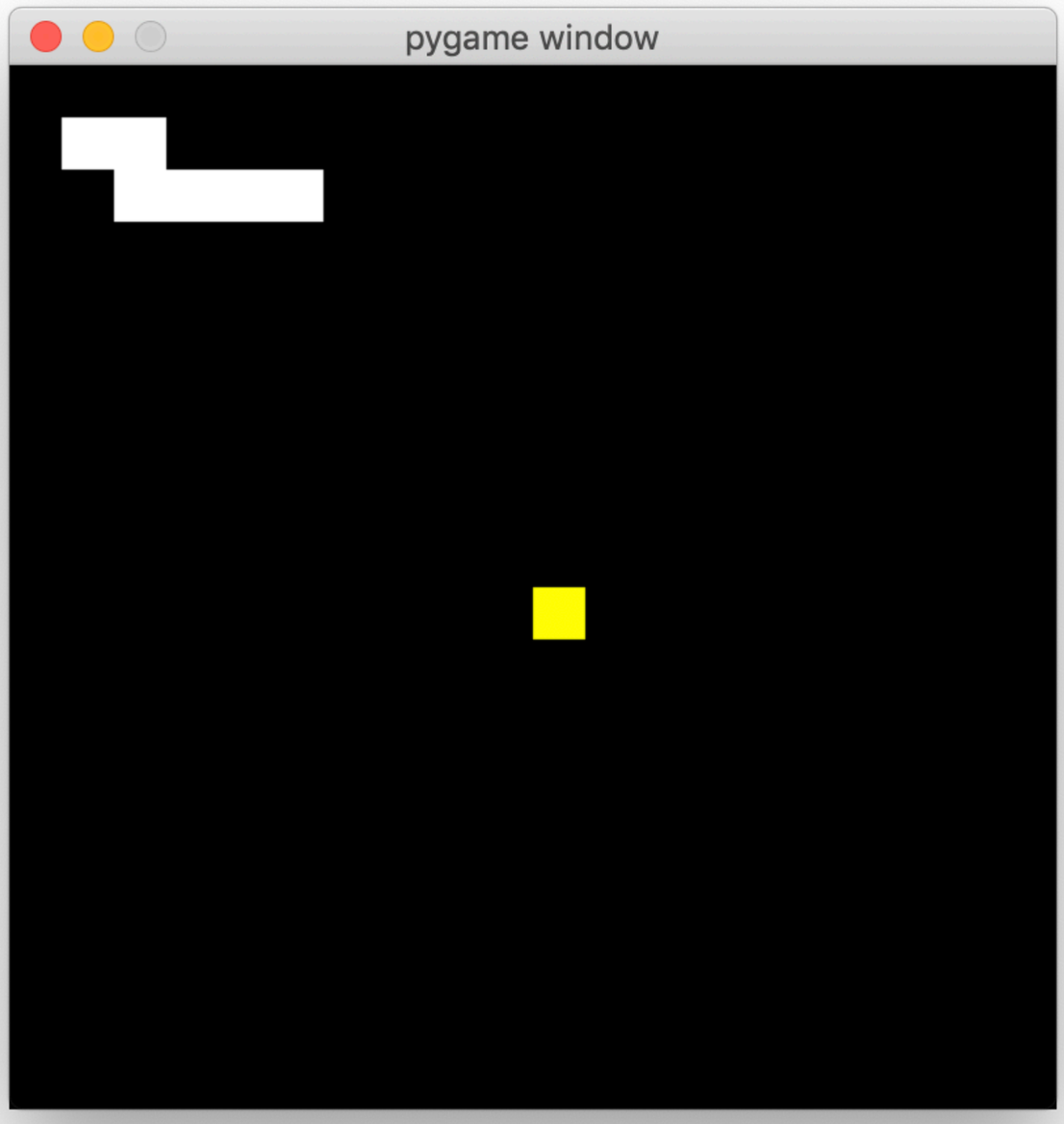


```
for pos in snake:
    draw_cell(pos, (255, 255, 255))
```


COMMIT

Exrcice: l'oeuf (super facile)

ajouter la position de l'oeuf à l'écran et l'afficher en jaune



```
egg_position = (10, 10)
#...

while True:
    #...
    draw_cell(egg_position, (255, 255, 0))
```


COMMIT
vous aviez deniné ?


otfilms
Animons

La notion de temps

```
clock = pygame.time.Clock()
#...

while True:
    delta_ms = clock.tick(60)
    #...
```

La notion de temps

```
clock = pygame.time.Clock()
#...

while True:
    delta_ms = clock.tick(60)
    #...
```

Exercice: le serpent qui bouge

Écrire une fonction `move_snake(dx, dy)` qui déplace le serpent de `dx` cases horizontalement et `dy` verticalement.

- → le serpent bouge **par la tête**, le reste du corps "suit"
- → *BONUS*: quand il arrivee au bout de l'écran, il ressort de l'autre côté

```python
def move_snake(dx, dy):
    x, y = snake[-1] # head
    new_head = (x+dx, y+dy)
    snake.append(new_head)
    snake.pop(0)
```

```python
BOARD_SIZE = (40, 40)

def move_snake(dx, dy):
    x, y = snake[-1]
    new_head = (
        (x+dx)%BOARD_SIZE[0],
        (y+dy)%BOARD_SIZE[1],
    )
    snake.append(new_head)
    snake.pop(0)
```


COMMIT

Exercice: on ralentit

le serpent bouge **trop vite**: modifier le programme pour qu'il ne se déplace que toutes les 500ms

```
time_since_last_update_ms = 0
#...

while True:
  time_since_last_update_ms += clock.tick(60)
  #...

  if time_since_last_update_ms > 500:
    time_since_last_update_ms = 0
    #...

    move_snake(dx, dy)
```

```
time_since_last_update_ms = 0
#...

while True:
  time_since_last_update_ms += clock.tick(60)
  #...

  if time_since_last_update_ms > 500:
    time_since_last_update_ms = 0
    #...

    move_snake(dx, dy)
```

```
time_since_last_update_ms = 0
#...

while True:
  time_since_last_update_ms += clock.tick(60)
  #...

  if time_since_last_update_ms > 500:
    time_since_last_update_ms = 0
    #...

    move_snake(dx, dy)
```


COMMIT

Évênements

contrôler le jeu

`pygame.locals` définit des valeurs de `event.key` pour les flèches dirctionnelles:

- la flèche du haut: `K_UP`
- la flèche du bas: `K_DOWN`
- la flèche de gauche: `K_LEFT`
- la flèche de droite: `K_RIGHT`

Exercice: déplacer le serpent

Compléter le programme pour détecter ces touches et déplacer le serpent en fonction

```python
# déplacement initial
dx, dy = 1, 0
#...

while True:
  #...
  for event in pygame.event.get(KEYDOWN):
    if event.key == K_q: # pygame.locals.K_q
      sys.exit() # quitte le programme
    elif event.key == K_UP:
      dx, dy = 0, -1
    elif event.key == K_DOWN:
      dx, dy = 0, 1
elif event.key == K_LEFT:
  dx, dy = -1, 0
elif event.key == K_DOWN:
  dx, dy = 1, 0
```


COMMIT

BONUS: en option

le serpent n'est pas autorisé à *revenir sur ces pas*, c-à-d.

- → repartir vers la gauche quand il va vers la droite
- → repartir vers la droit quand il va vers la gauche
- → repartir vers le haut quand il va vers le bas
- → repartir vers la bas quand il va vers le haut

modifier le programme pour éviter ces cas

Finalisation

(gagner) et perdre

Exercice: l'oeuf aléatoire

créer une fonction `place_egg()` qui crée un oeuf à un position aléatoire du plateau, **non occupée par le serpent**

- lire la doc de `random.randrange()`
- appeler la méthode en début de progamme pour placer le premier oeuf

```
from random import randrange

def place_egg():
  max_x, maxY = BOARD_SIZE
  return (randrange(0, max_x), randrange(0, max_y))
```

```
from random import randrange

def place_egg():
    max_x, maxY = BOARD_SIZE
    return (randrange(0, max_x), randrange(0, max_y))
```

```
from random import randrange

def place_egg():
  max_x, maxY = BOARD_SIZE
  return (randrange(0, max_x), randrange(0, max_y))
```


COMMIT

Exercice: manger l'oeuf

Modifier le code pour que lorsque le serpent mange un oeuf, il se ralonge de une cellule

```python
# le serpent mange "par la tête"
def can_eat():
  head = snake[-1]
  return pos == head
#...

def move_snake(dx, dy, eats=False):
  if not eats:
    snake.pop(0)
  x, y = snake[-1]
  new_head = (x+dx, y+dy)
  snake.append(new_head)

while True:
  #...
  eats = can_eat()
  move_snake(dx, dy, eats)
  if eats:
    place_egg()
```

```python
# le serpent mange "par la tête"
def can_eat():
  head = snake[-1]
  return pos == head
#...

def move_snake(dx, dy, eats=False):
  if not eats:
    snake.pop(0)
  x, y = snake[-1]
  new_head = (x+dx, y+dy)
  snake.append(new_head)

while True:
  #...
  eats = can_eat()
  move_snake(dx, dy, eats)
  if eats:
    place_egg()
```

```python
# le serpent mange "par la tête"
def can_eat():
  head = snake[-1]
  return pos == head
#...

def move_snake(dx, dy, eats=False):
  if not eats:
    snake.pop(0)
  x, y = snake[-1]
  new_head = (x+dx, y+dy)
  snake.append(new_head)

while True:
  #...
  eats = can_eat()
  move_snake(dx, dy, eats)
  if eats:
    place_egg()
```


COMMIT

Exercice: le serpent se mord la queue

créer une fonction `end_of_game()` qui détecte si le serpent se "mord la queue", c-à-d. si ça tête arrive sur un case déjà occupée par son corps.

```
def end_of_game():
    head = snake[-1]
    return any(pos == head for pos in snake[:-1])
```


COMMIT

Partie II: get real

Transformer ce code en un vrai projet Python avec des classes et tout...

Créons une classe

Dans sa version actuelle, notre programme manque d'organisation.

Commençons par embarquer notre projet à l'intérieur d'une classe `Game` qui encapsule le code actuel.

Exercice

Regrouper tout le code du jeu à l'intérieur d'une classe `Game`, telle que le démarrage du jeu se fasse ainsi:

```
game = Game(
  board_size=(40, 40),
  cell_size=(20, 20)
)
game.start()
```

```
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
YELLOW = (255, 255, 0)
class Game:
    def __init__(self, board_size, cell_size):
        self.board_size = board_size
        self.cell_size = cell_size
        window_size = (self.board_size[0]*self.cell_size[0], self.board_size[1]*self.cell_size[1])
        self.screen = pygame.display.set_mode(window_size)
        self.clock = pygame.time.Clock()
        self.time_since_last_update_ms = 0
        self.dx = 1
        self.dy = 0
        self.snake = [(1,0), (2,0)]
        self.egg = (10, 10)
    def start(self):
        while not self.end_of_game():
            self.time_since_last_update_ms += self.clock.tick(60)

            # 1. poll controls
            self.poll_keys()
            # 2. update state
            if self.time_since_last_update_ms > 200:
                self.time_since_last_update_ms = 0
                eats = self.can_eat()
    self.move_snake(eats)
    if eats:
        self.place_egg()
# 3. draw
self.screen.fill(BLACK)
self._draw_cell(self.egg, YELLOW)
for pos in self.snake:
    self._draw_cell(pos, WHITE)
pygame.display.update()
```

```
def poll_keys(self):
    for event in pygame.event.get(KEYDOWN):
        if event.key == K_q:
            sys.exit()
        elif event.key == K_UP:
            self.dx, self.dy = 0, -1
        elif event.key == K_DOWN:
            self.dx, self.dy = 0, 1
        elif event.key == K_LEFT:
            self.dx, self.dy = -1, 0
        elif event.key == K_RIGHT:
            self.dx, self.dy = 1, 0
def can_eat(self):
    return self.egg == self.snake[-1]
def move_snake(self, eats):
    x, y = self.snake[-1]
    new_head = ((x+self.dx) % self.board_size[0], (y+self.dy) % self.board_size[1])
    self.snake.append(new_head)
    if not eats:
        self.snake.pop(0)
def place_egg(self):
    w, h = self.board_size
    available_cells = [pos for pos in product(range(w), range(h)) if pos != self.egg and pos not in self.snake]
    self.egg = choice(available_cells)
def end_of_game(self):
    head = self.snake[-1]
    return any(pos == head for pos in self.snake[:-1])
def _draw_cell(self, board_pos, color):
    x, y = board_pos
    cell_width, cell_height = self.cell_size
    for line in range(cell_height):
        for col in range(cell_width):
            screen_pos = (cell_width * x + col, cell_height * y + line)
            self.screen.set_at(screen_pos, color)
```

Créons un module et un package

Comme nous l'avons vu pendant le **Python Primer**:

- un module est un fichier `.py`
- un package est un répertoire contenant des modules