Compte rendu technique

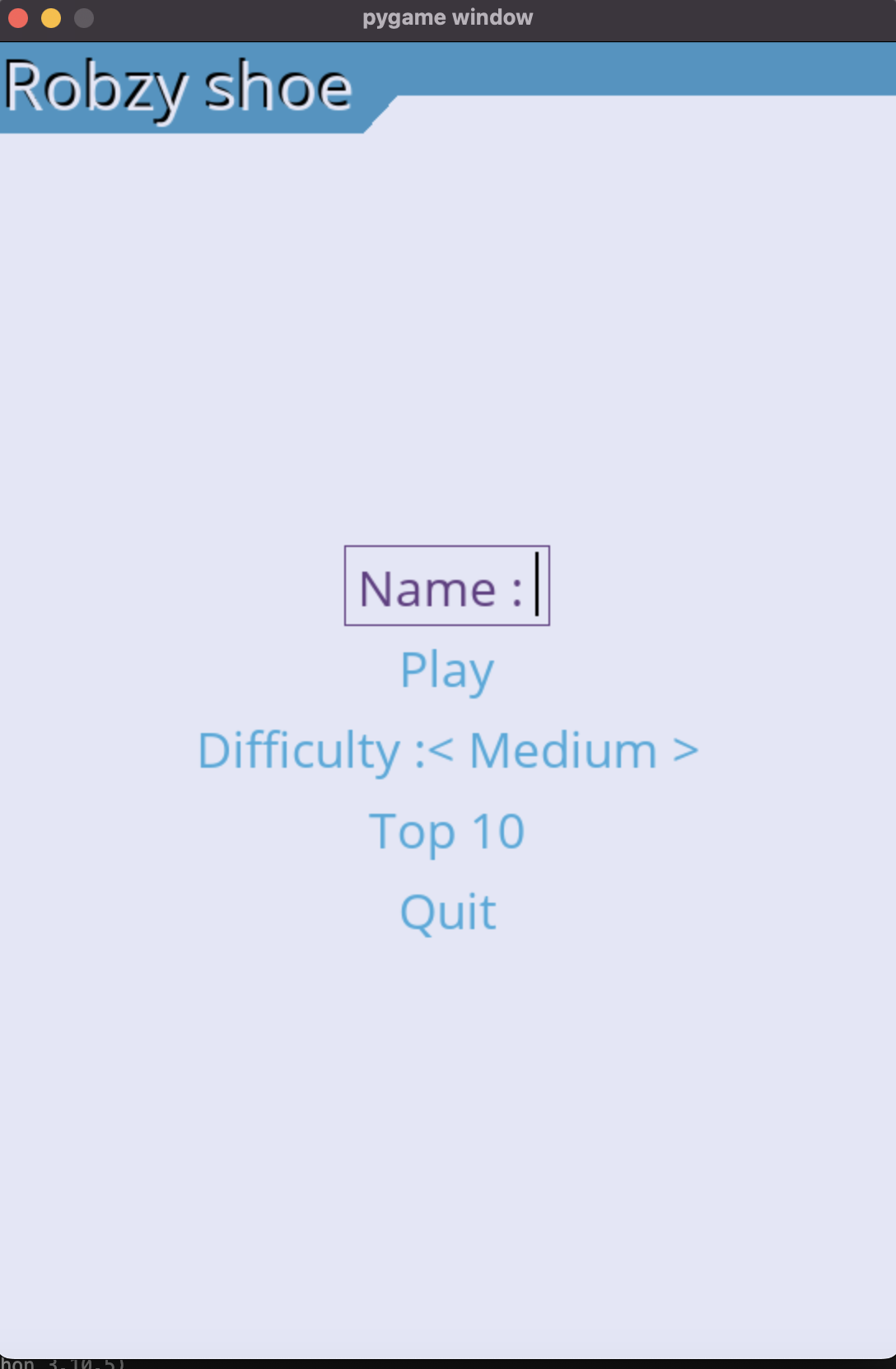
# Présentation :

L'application développée est un jeu Flappy Bird Like : Personnage qui se déplace sur un plan 2d et qui doit passer à travers des obstacles, le jeu s'arrête une fois que le joueur touche soit un obstacle, soit va trop loin dans le ciel soit touche le sol. A chaque fois que le personnage passe une rangée d'obstacle alors le score est incrémenté de 1.

Avant de jouer, le joueur à accès à un menu ou il doit rentrer son nom, il peut aussi voir le top 10 des scores qui ont été fait et peut télécharger un pdf avec l'historique des anciennes parties et des informations d'agrégation type : score moyen…

# Installation :

Pour lancer le programme, il faut une install python 3.x.x avec les librairies :

* pygame
* pygame\_menu
* os
* sys
* time
* random
* sqlite3

Pouvant être fait avec le requirements .txt

pip install -r requirements.txt

# Mode d'emploi :

## Le menu :

Voici le menu ou l'on peut jouer,

voir les 10 meilleurs scores par difficulté

et rentrer son nom pour se repérer dans les scores.

## 

## Le jeu en lui même :

Ici le joueur va contrôler la position en Y de la

chaussure jaune verte et bleue.

Elle est soumise à la gravité et le joueur pourra la

faire "Sauter" en pressant la touche espace.

Les obstacles à éviter sont les piles de chaussures.

Mais il faut faire attention à ne pas toucher le sol et à

ne pas aller trop car il y a une limite (pour éviter la

triche).

Plus la difficulté est élevé, plus les piles de

chaussures seront rapprochées

## Les meilleurs scores :

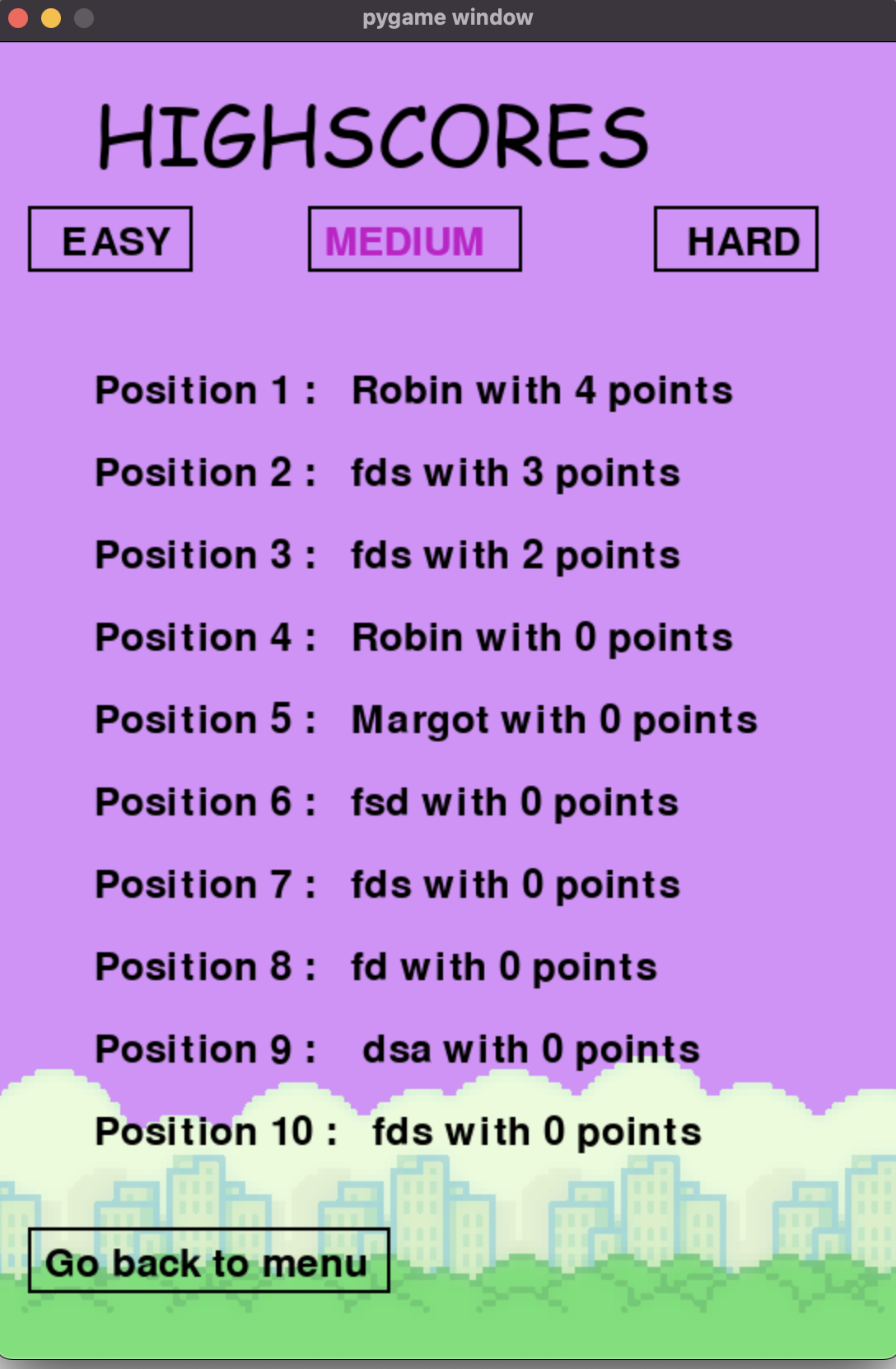
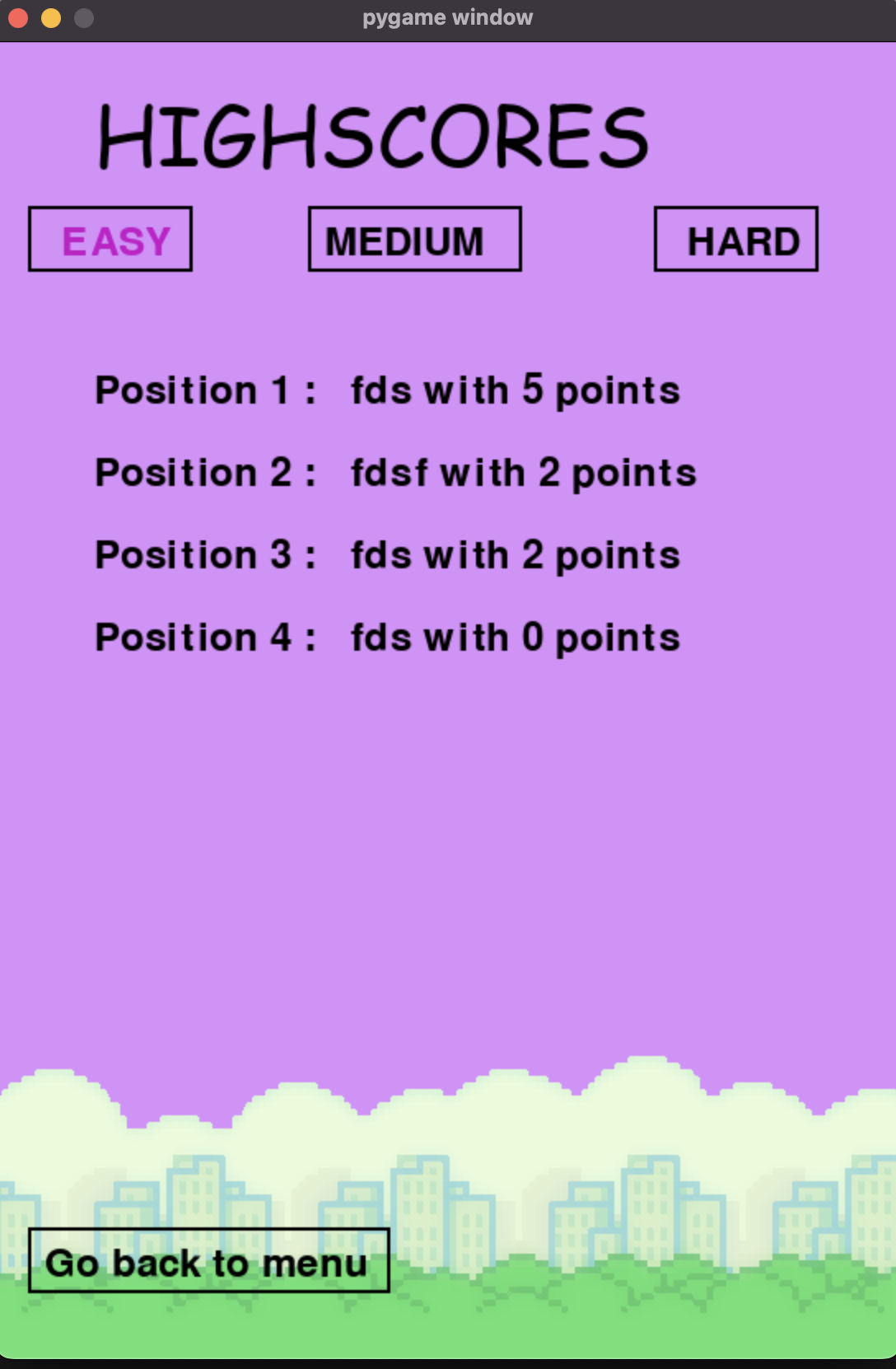
Voici l'affichage depuis la base de données sqlite3

des meilleurs scores.

Pour ce faire on fait une requêtes en triant de

manière décroissante les scores et en limitant à 10

le nombre de lignes en sortie.



Conclusion technique :

Nous avions tous les deux déjà utilisé python mais nous sommes novices et n'avons pas plus approfondie que ça la librairie pygame qui permet l'interface graphique. Nous avions plus envie d'approfondir les mécaniques du jeu et d'appliquer des vraies lois de physique pour obtenir de vraies paraboles lors des sauts. Pour cela nous avons calculé la différence entre le y de la chaussure à la frame f et le y de la chaussure à la frame f+1.

Alors pour la partie menu nous avons utilisé la librairie pygame\_menu qui permet d'avoir une interface sobre et efficace et qui répond correctement aux inputs de l'utilisateur :

* Click à la souris
* Flèches directionnelles
* Touche entrée

Nous avons pour enregistrer de la données utilisé une base de données sqlite3. Histoire de pouvoir enregistrer les différentes informations essentielles :

* Table : records
  + temps
  + score
  + difficulté
  + date
  + nom\_joueur
  + id\_record

# Code Robin :

La partie que je tiens à présenter est le calcul de la nouvelle position à chaque frame.

Notre chaussure que l'on contrôle est un objet Shoe définit comme tel :

class Shoe:

IMG = SHOE\_IMG

MAX\_ROTATION = 25

ROT\_VEL = 10

ACCELERATION = 3

velocity = 0

def \_\_init\_\_(self):

self.x = 230

self.y = 360

self.rota = 0

self.time = 0

self.height = self.y

self.img = self.IMG

Velocity est en fait la vélocité sur l'axe y est égale à -10 car l'échelle est inversée avec pygame donc d'ou le moins. Mais au démarrage il faut que la chaussure tombe et donc au premier saut, on va changer la valeur de cette vitesse (Commencement 0 puis après -10).

def jump(self):

# Echelle par du corner en haut à gauche donc il nous faut une vel négative

self.time = 0

Shoe.velocity = -10

# Position ou on débute notre saut pour analyser la descente plus tard

self.height = self.y

On sait que :

Nouvelle position = ancienne position + velocité \* temps + 1/2 \* acceleration \* temps^2

Pour ne pas trop compliquer la tâche, nous avons appliqué une accélération et une vitesse constante. La différence de position va donc se voir par rapport au temps qui sera la seule variable qui changera de valeur.

- On va décomposer l'équation en deux partie :

* velocité \* temps sera toujours négatif à cause de notre var velocity = -10 et donc cette partie de l'équation va pousser la chaussure vers le haut
* 1/2 \* acceleration \* temps^2 sera nul à la première frame, sera ensuite faible mais à cause de l'exposant 2 sera plus grand que la première partie lorsque plusieurs frames auront défilées.

Le résultat est que notre chaussure va faire une parabole contre la gravité à chaque saut car notre fonction saut va reset le temps à 0. Enfin on ajoute cette valeur à notre ancienne position.

def move(self):

self.time +=1

# Lois de newton : https://www.youtube.com/watch?v=v\_linpA7uXo&ab\_channel=CDcodes

# Nouvelle position = ancienne position + velocité \* temps + 1/2 \* acceleration \* temps^2

d = Shoe.velocity \* self.time + 0.5 \* Shoe.ACCELERATION \* self.time\*\*2

# On set une distance entre 2 frames différentes car sinon c'est incontrolable

# On doit passer par là car on veut une gravité relativemment faible pour le jeu

if d >= 15:

d = 15

self.y = self.y + d

Enfin on limite le déplacement en 2 frames à 15 pixel car sinon au bout d'un certain nombre de frames, la chaussure devient incontrôlable.

C'était intéressant de faire des animations plus poussées que sur la plupart des jeux que j'ai essayé de coder. C'était compliqué donc je me suis pas mal inspiré sur le web mais au final le rendu est beaucoup plus propre.

J'aurai aussi aimé présenter la partie collision, car grâce à pygame on peut délimiter notre chaussure non pas par sa largeur/longueur (brute). Il y a une fonction qui permet de trouver les limites en ne prenant pas en compte les pixels transparents.

# On créer un masque pour checké les pixel à l'intérieur de nos hitbox pour que les collisions soit on point

def collide(self,shoe):

shoe\_mask = shoe.mask()

top\_mask = pygame.mask.from\_surface(self.PILE\_TOP)

bottom\_mask = pygame.mask.from\_surface(self.PILE\_BOTTOM)

top\_offset = (self.x - shoe.x,self.top - round(shoe.y))

bottom\_offset = (self.x - shoe.x, self.bottom - round(shoe.y))

b\_point = shoe\_mask.overlap(bottom\_mask,bottom\_offset)

t\_point = shoe\_mask.overlap(top\_mask,top\_offset)

if b\_point or t\_point:

return True

else :

return False

Overlap va donc tester si les masques de la chaussures et le masque de la pile ou de la base s'entrecroise.

Si oui alors b\_point et t\_point seront vrais et donc la collision sera repérer

def mask(self):

return pygame.mask.from\_surface(self.img)

# Code Thibault :

Les 2 parties principales étaient la gestion de la donnée et la création du menu.

Pour la base de données j'ai utilisé la base de données embed sqlite3.

import sqlite3

import datetime

global con

con = sqlite3.connect("fuckJ1.db")

global c

c = con.cursor()

def getScoresDesc(difficulty):

c.execute("Select \* from records WHERE difficulty = ? order by score desc limit 10",(difficulty,))

return c.fetchall()

def insertScore(player\_name, score,jump\_count,time,difficulty):

date\_str = str(datetime.datetime.now())[:10] +" "+ str(datetime.datetime.now())[11:16]

c.execute("insert into records(score,player\_name,date\_hour,jump\_count,time,difficulty) values (?, ?, ?, ?, ?, ?)",

(score, player\_name, date\_str,jump\_count,time,difficulty))

con.commit()

def returnRecordsDate(date):

c.execute("Select \* from records")

score = c.fetchall()

for row in score:

x = row[3][0:10]

if x == date:

print(row)

La fonction getScoreDesc( difficulté ) permet de requêter la BDD comme son nom le dit avec en paramètre la difficulté. Le but de cette fonction est l'affichage des top 10 scores avec: le score, le nom.

Donc ici je récupère tout puis je n'affiche et ne choisit que les informations importantes dans le main.

La fonction insertScore permet à chaque fin de partie l'insertion du score dans la base de données mais cette fois avec beaucoup plus de paramètres. Nous voulions récolter plein d'informations pour faire du traitement de données mais par manque de temps et d'intérêt nous avons préféré abandonner.

La fonction returnReocrdsDate permet de ne récupérer les enregistrements seulement si la date passé en paramètre est bonne. Après notre cours de base de données relationnelles, nous nous sommes dit quand pour plus tard il serait possible de mettre des procédures directement et de les appeler dans le code.

def player\_name(name):

global PLAYER\_NAME

PLAYER\_NAME = name

def set\_difficulty(value, difficulty):

global DIFFICULTY

DIFFICULTY = value

def display\_menu():

def startGame():

if test.get\_value() == "":

pass

else:

menu.disable()

start\_the\_game()

def highScore():

menu.disable()

seeHighScore()

win\_menu = pygame.display.set\_mode((WINDOW\_WIDTH,WINDOW\_HEIGHT))

menu = pygame\_menu.Menu('FUCK J1\'S', 550, 800,theme=pygame\_menu.themes.THEME\_BLUE)

test=menu.add.text\_input('Name :',onchange=player\_name,default=" ")

menu.add.button('Play', startGame)

menu.add.selector('Difficulty :', [('Hard', 1), ('Medium',2),('Easy', 3)], onchange=set\_difficulty, default=1)

menu.add.button('Top 10', highScore)

menu.add.button('Quit', pygame\_menu.events.EXIT)

menu.mainloop(win\_menu)

def main():

display\_menu()

Et voici la partie qui génère le menu avec la librairie pygame-menu. On peut donc voir que c'est assez simple à lire comme code. On génère notre menu dans notre window et générons toutes les différentes lignes qui laissent la possibilité de récupérer des informations tel que la difficulté, le nom, de lancer la partie ou de quitter.