Snake sur raspberry pi

Renaud TAHON – Florian SIMOND

Haute Ecole en Hainaut – Campus Technique

Initiation aux nano-contrôleurs

Table des matières

[1. Introduction 3](#_Toc482381483)

[2. Choix techniques 4](#_Toc482381484)

[2.1. Choix du langage de programmation 4](#_Toc482381485)

[2.2. Choix de l’IDE 4](#_Toc482381486)

[2.3. Choix du système d’exploitation (OS) 4](#_Toc482381487)

[3. Algorithme de programmation 5](#_Toc482381488)

[4. Problèmes rencontrés 14](#_Toc482381489)

[4.1. Boot sur la carte SD 14](#_Toc482381490)

[4.2. Connection SSH au Raspberry Pi 14](#_Toc482381491)

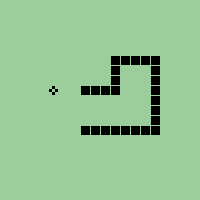
[4.3. Buffer d’événements 14](#_Toc482381492)

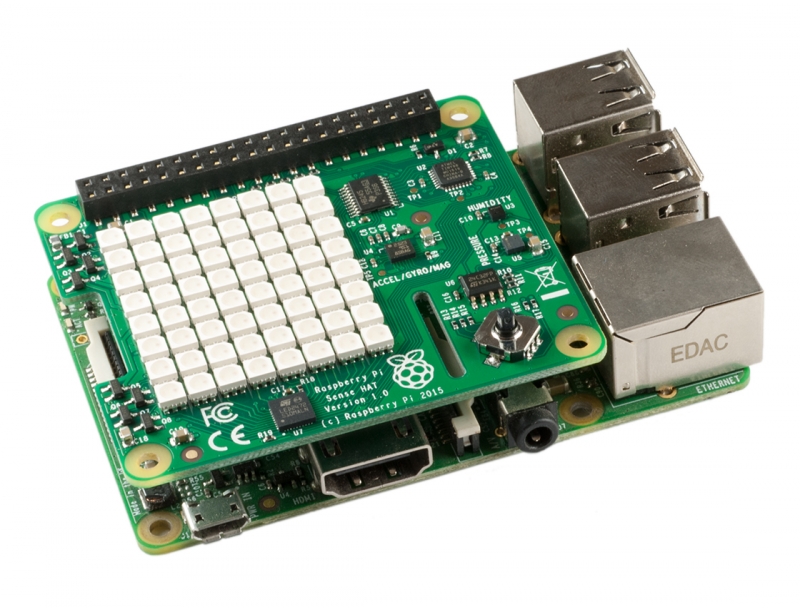
[5. Conclusion 15](#_Toc482381493)

# Introduction

Notre projet consiste en la réalisation du jeu « Snake » sur le Rapsberry Pi et plus spécialement sur le module « Sense Hat » avec une gestion de la difficulté par augmentation de la vitesse de déplacement du Snake et avec les contrôles sur le joystick ainsi qu’en utilisant la rotation du Raspberry.

Pour cela, nous nous sommes renseigné sur les règles du Snake, la documentation sur le Rapsberry Pi et sur le Sense Hat.





# Choix techniques

## Choix du langage de programmation

Nous avons décidé d’utiliser le langage de programmation Python, car c’est ce langage qui est le plus utilisé pour le Raspberry Pi, de plus toute la documentation mise à notre disposition était en Python.

## Choix de l’IDE

Notre choix s’est porté sur un IDE en ligne : pep8online.

Dans un souci de facilité, nous avons décidé de faire la programmation en dehors du Raspberry Pi, et d’utiliser un émulateur de Sense Hat pour tester notre code. Ce qui nous a permis de travailler sur le code aussi en dehors des heures consacrées au projet dans notre horaire.

Nous avons donc codé toute l’application sur notre ordinateur et nous avons vérifié le code grâce à l’IDE en ligne, puis lorsque nous devions tester la compatibilité avec le Raspberry Pi, nous mettions le code sur celui-ci.

## Choix du système d’exploitation (OS)

Nous avons choisi « Raspbian », car c’est un dérivé de Debian, une distribution Linux.

Nous avons eu un cours sur Linux au premier quadrimestre, ce qui nous a permis de déjà maitriser un peu cet OS.

De plus, cet OS a une forte compatibilité avec le Raspberry Pi.

# Algorithme de programmation

# --------------------------------- #

# --- Importation des libraries --- #

from sense\_hat import SenseHat

import time

import random

# --------------------------------- #

# --- Declaration des variables --- #

s = SenseHat()

#s.set\_imu\_config(False, True, False)

last\_direction = 'right'

red = (255, 0, 0)

white = (255, 255, 255)

green = (0, 255, 0)

nothing = (0, 0, 0)

listX = [3, 2, 1, 0]

listY = [3, 3, 3, 3]

appleX = 0

appleY = 0

score = 0

level = 1

appleCount = 0

#Variables pour gyroscope

'''

global orientation

orientation = s.get\_orientation()

global left

left = False

global leftOr

leftOr = 0

global right

right = False

global rightOr

rightOr = 0

global top

top = False

global topOr

topOr = 0

global bottom

bottom = False

global bottomOr

bottomOr = 0

global playModeRoll

playModeRoll = True

'''

# Variable de vérification

global on

on = True

def getOn():

return on

def setOn(temp):

global on

on = temp

# --------------------------------- #

# ----------- Fonctions ----------- #

# Fonction pour l'affichage du snake et des pommes

def update\_screen():

s.clear()

s.set\_pixel(listX[0], listY[0], white)

s.set\_pixel(listX[1], listY[1], white)

s.set\_pixel(listX[2], listY[2], white)

s.set\_pixel(listX[3], listY[3], white)

s.set\_pixel(appleX, appleY, red)

# Fonction de génération aléatoire de pommes ( une à la fois )

def setup\_apple():

global appleX

global appleY

posOK = False

while not posOK:

appleX = random.randint(0, 7)

appleY = random.randint(0, 7)

i = 0

posOK = True

while i < 4:

if listX[i] == appleX and listY[i] == appleY:

posOK = False

i += 1

# Fonction de vérification si une pomme a été mangée

def check\_apple():

global score

global appleCount

global level

if listX[0] == appleX and listY[0] == appleY:

score += 1\*level

appleCount += 1

if appleCount % 2 == 0:

level += 1

setOn(False)

s.show\_message("Level " + str(level), 0.05, green)

s.stick.get\_events().clear()

update\_screen()

setup\_apple()

setOn(True)

# Fonction qui met à jour la position du snake sur le tableau de pixels

def move\_body():

listX[3] = listX[2]

listX[2] = listX[1]

listX[1] = listX[0]

listY[3] = listY[2]

listY[2] = listY[1]

listY[1] = listY[0]

# Fonction liée aux événements du joystick

def move(event):

global appleCount

if getOn():

if event.action in ('pressed', 'held'):

global last\_direction

if event.direction == 'up': # Avance vers le haut

if last\_direction != 'down':

if listY[0] > 0:

move\_body() # Déplacement du snake

listY[0] = listY[0] - 1

check\_apple() # Vérification si une pomme a été mangée

last\_direction = event.direction # Pour le move auto

else:

game\_over()

elif event.direction == 'down': # Avance vers le bas

if last\_direction != 'up':

if listY[0] < 7:

move\_body()

listY[0] = listY[0] + 1

check\_apple()

last\_direction = event.direction

else:

game\_over()

elif event.direction == 'left': # Avance vers la gauche

if last\_direction != 'right':

if listX[0] > 0:

move\_body()

listX[0] = listX[0] - 1

check\_apple()

last\_direction = event.direction

else:

game\_over()

elif event.direction == 'right': # Avance vers la droite

if last\_direction != 'left':

if listX[0] < 7:

move\_body()

listX[0] = listX[0] + 1

check\_apple()

last\_direction = event.direction

else:

game\_over()

# Fonction qui gère le déplacement automatique du snake

def move\_auto():

if last\_direction == 'left': # Avance automatiquement vers la gauche

if listX[0] > 0:

move\_body()

listX[0] = listX[0] - 1

check\_apple()

update\_screen()

else:

game\_over()

elif last\_direction == 'right': # Avance automatiquement vers la droite

if listX[0] < 7:

move\_body()

listX[0] = listX[0] + 1

check\_apple()

update\_screen()

else:

game\_over()

elif last\_direction == 'up': # Avance automatiquement vers le haut

if listY[0] > 0:

move\_body()

listY[0] = listY[0] - 1

check\_apple()

update\_screen()

else:

game\_over()

elif last\_direction == 'down': # Avance automatiquement vers le bas

if listY[0] < 7:

move\_body()

listY[0] = listY[0] + 1

check\_apple()

update\_screen()

else:

game\_over()

#Fonctions pour mouvements avec gyroscope

'''

def check():

if orientation['pitch'] > 300 and orientation['pitch'] < 345:

#droite

global right

global rightOr

right = True

rightOr = 345 - orientation['pitch']

if orientation['pitch'] > 15 and orientation['pitch'] < 60:

#gauche

global left

global leftOr

left = True

leftOr = 15 + orientation['pitch']

if orientation['roll'] > 300 and orientation['roll'] < 345:

#haut

global top

global topOr

top = True

topOr = 345 - orientation['roll']

if orientation['roll'] > 15 and orientation['roll'] < 60:

#bas

global bottom

global bottomOr

bottom = True

bottomOr = 15 + orientation['roll']

doTheMove()

def doTheMove():

if left:

if top:

if leftOr > topOr:

if listX[0] > 0:

move\_body()

listX[0] = listX[0] - 1

check\_apple()

update\_screen()

last\_direction = 'left'

else:

game\_over()

elif topOr >= leftOr:

if listY[0] > 0:

move\_body()

listY[0] = listY[0] - 1

check\_apple()

update\_screen()

last\_direction = 'up'

else:

game\_over()

elif bottom:

if leftOr > bottomOr:

if listX[0] > 0:

move\_body()

listX[0] = listX[0] - 1

check\_apple()

update\_screen()

last\_direction = 'left'

else:

game\_over()

elif bottomOr >= leftOr:

if listY[0] < 7:

move\_body()

listY[0] = listY[0] + 1

check\_apple()

update\_screen()

last\_direction = 'down'

else:

game\_over()

else:

if listX[0] > 0:

move\_body()

listX[0] = listX[0] - 1

check\_apple()

update\_screen()

last\_direction = 'left'

else:

game\_over()

elif right:

if top:

if rightOr > topOr:

if listX[0] > 0:

move\_body()

listX[0] = listX[0] - 1

check\_apple()

update\_screen()

last\_direction = 'right'

else:

game\_over()

elif topOr >= rightOr:

if listY[0] > 0:

move\_body()

listY[0] = listY[0] - 1

check\_apple()

update\_screen()

last\_direction = 'up'

else:

game\_over()

elif bottom:

if rightOr > bottomOr:

if listX[0] > 0:

move\_body()

listX[0] = listX[0] - 1

check\_apple()

update\_screen()

last\_direction = 'right'

else:

game\_over()

elif bottomOr >= rightOr:

if listY[0] < 7:

move\_body()

listY[0] = listY[0] + 1

check\_apple()

update\_screen()

last\_direction = 'down'

else:

game\_over()

else:

if listX[0] > 0:

move\_body()

listX[0] = listX[0] - 1

check\_apple()

update\_screen()

last\_direction = 'right'

else:

game\_over()

elif top:

if listY[0] > 0:

move\_body()

listY[0] = listY[0] - 1

check\_apple()

update\_screen()

last\_direction = 'up'

else:

game\_over()

elif bottom:

if listY[0] < 7:

move\_body()

listY[0] = listY[0] + 1

check\_apple()

update\_screen()

last\_direction = 'down'

else:

game\_over()

else:

move\_auto()

'''

# Fonction de fin de jeu

def game\_over():

global last\_direction

global score

global level

global appleCount

s.clear()

setOn(False)

s.show\_message("GAME OVER: " + str(score) + " points", 0.05, red)

setOn(True)

# Réinitialisation des variables pour une nouvelle partie

listX[3] = 0

listX[2] = 1

listX[1] = 2

listX[0] = 3

listY[3] = 3

listY[2] = 3

listY[1] = 3

listY[0] = 3

last\_direction = 'right'

setup\_apple()

score = 0

level = 1

appleCount = 0

# Fonction principale qui fait tourner le jeu et qui gère la difficulté

def run():

global orientation

while True:

while getOn():

move\_auto()

if level < 34:

time.sleep(2 - (level \* 0.1))

else:

time.sleep(0.3)

#Boucle pour jouer avec le gyroscope

'''

while playModeRoll:

check()

orientation = s.get\_orientation()

left = False

right = False

top = False

bottom = False

if level < 34:

time.sleep(2 - (level \* 0.1))

else:

time.sleep(0.3)

'''

time.sleep(1)

# --------------------------------- #

# ---- Gestion des événements ----- #

s.stick.direction\_up = move

s.stick.direction\_down = move

s.stick.direction\_left = move

s.stick.direction\_right = move

s.stick.direction\_any = update\_screen

# --------------------------------- #

# --- Appel des fonctions de jeu ---#

setup\_apple()

update\_screen()

run()

# Problèmes rencontrés

## Boot sur la carte SD

Lors de la première séance, il nous était impossible de faire booter notre OS, sur le Rapsberry Pi, nous avons donc essayé un autre logiciel de formatage de la carte SD (Etcher), et le problème était résolu.

## Connection SSH au Raspberry Pi

Lors des séances, il nous a fallu 12h de labo pour nous connecter en SSH sur le Rapsberry Pi, il nous était impossible de trouver son adresse IP. Nous avons testé plusieurs méthodes, reformaté la carte, changé de Rapsberry Pi, essayé une autre carte, essayé un autre OS, mais rien ne fonctionnait.

Nous avons résolu ce problème en entrant l’adresse IP directement dans un fichier (cmdline.txt), de ce fait, nous avons imposé une adresse IP au Raspberry Pi, et nous nous y sommes connecté avec PuTTy, et le problème était résolu.

## Buffer d’événements

Lors de l’affichage d’un message sur l’écran LED du Sense Hat, nous avions un problème, dans certains cas, la matrice de LED faisait une rotation et de plus lorsqu’on appuyait sur le joystick du Sense Hat, les événements étaient gardés dans une pile et ils s’exécutaient une fois le message affiché entièrement. Ce cas n’apparaissait que lors d’une situation particulière : l’appel à des événements durant l’affichage du message.

Pour résoudre ce souci, nous avons cherché dans les fin fonds de l’API du Sense Hat et nous avons trouvé une fonction pour gérer la liste des événements dans la pile, et il nous a suffi de vider cette pile juste après avoir fait l’affichage du message.

# Conclusion

Malgré plusieurs soucis de programmation comme dans tous programmes, nous avons fini notre projet, le Snake est fonctionnel avec le joystick totalement. Concernant la seconde partie, c’est-à-dire le contrôle du Snake par rotation du Raspberry, n’ayant pas eu assez de temps, nous n’avons pas su totalement le finir. Tous les déplacements sauf vers le bas sont gérés (le gyroscope ne retourne pas de valeurs correctement utilisables, c’est-à-dire la même valeur qu’on le tourne vers le haut ou vers le bas : valeur retournée par la variable ‘pitch’).

Dans l’ensemble, nous avons rencontré pas mal de problèmes mais nous avons tous su les résoudre. Le principal problème était le langage, nous ne le maitrisions pas avant la réalisation du projet.

Dans la perspective d’amélioration du projet, il nous faudrait totalement finir la gestion des contrôles via la rotation du Raspberry, faire une interface graphique, voir même faire une gestion des différents scores et les stocker.