Projet

Jeu de Pong

2ème Bachelier en Informatique

Initiation aux nano-ordinateurs

Auteur :
Terencio Agozzino

Auteur :
Alexandre Ducobu

Enseignants :
Matthieu Michiels
David Arnaud

Année académique 2016 - 2017

Projet

Jeu de Pong

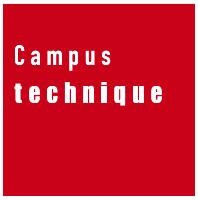
2ème Bachelier en Informatique

Initiation aux nano-ordinateurs

Auteur :
Terencio Agozzino

Auteur :
Alexandre Ducobu

Enseignants :
Matthieu Michiels
David Arnaud



Année académique 2016 - 2017

Nous remercions en particulier
Monsieur David ARNAUD de nous
avoir permis d'utiliser le matériel
en dehors des heures de laboratoire.

Ainsi que Monsieur Florian DI VRUSA
pour nous avoir indiqué une erreur
au niveau de la prise en charge de
la librairie Sense HAT en Python.

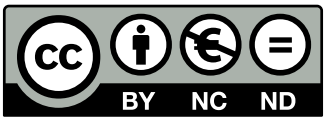


Table des matières

1 Présentation générale du projet

1.1 Introduction

Dans le cadre de ce projet, il nous a été demandé d'implémenter un jeu de Pong sur un Raspberry Pi 3.0 à l'aide d'un Sense HAT.

Vu la taille de la matrice du Sense HAT *(de 8 LED's par 8)*, notre jeu de Pong subira diverses modifications par rapport à la version officielle.

D'après les règles standards et les contraintes imposées, notre jeu de Pong comportera les éléments suivants :

— un joystick pour guider la raquette ;

— un code couleur se rapportant au logo de la HEH (*rouge et blanc*) ;

— l'affichage des points et des niveaux ;

— les rebonds de 45°.

1.2 Jeu de Pong

D'un point de vue historique, le jeu de Pong est le premier jeu vidéo d'arcade de sport créé. Celui-ci a été commercialisé en novembre 1972.

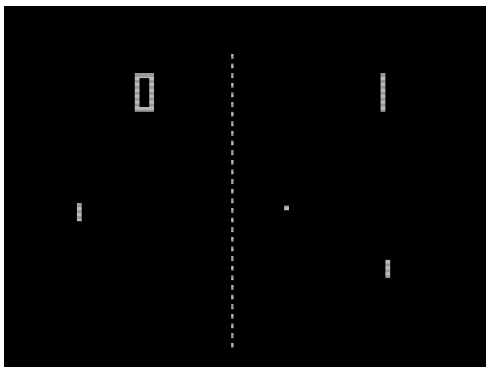


FIGURE 1 – Premier jeu de Pong

Ce jeu est constitué de deux raquettes et d'une balle. Celle-ci ne peut toucher que les murs supérieurs et inférieurs en effectuant un rebond selon un angle déterminé. Lorsque la balle atteint le mur de gauche ou de droite, la partie est terminée.

1.3 Modes de jeu

Notre version du jeu de Pong propose deux modes :

— Un premier, disponible lors des deux premiers niveau où le joueur joue contre un mur.

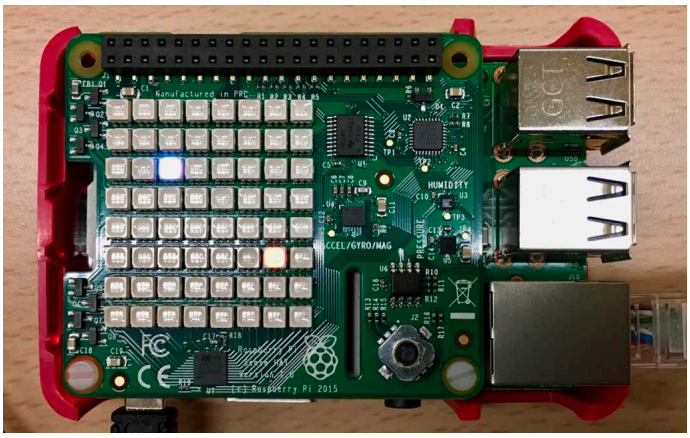


FIGURE 2 – Joueur contre mur

— Un deuxième, à partir du troisième niveau, permettant à une intelligence artificielle (IA) de faire son apparition dans le jeu, augmentant les similitudes avec le jeu de pong traditionnel.

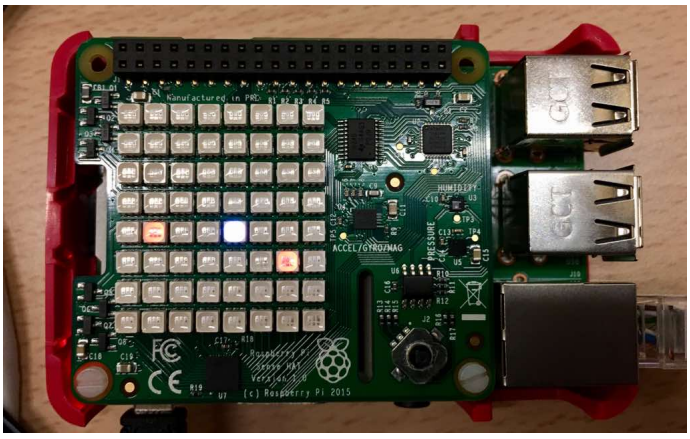


FIGURE 3 – Joueur contre IA

Remarque : la vitesse du jeu augmente progressivement à chaque jeu.
De même, elle augmente lors du passage au niveau supérieur.

1.4 Raspberry Pi

Le Raspberry Pi [1] est un nano-ordinateur équipé d'un microprocesseur ARM.

FIGURE 4 – Raspberry Pi, Modèle 3 B

En raison de sa très faible consommation en énergie, celui-ci est majoritairement utilisé en tant que serveur, puisqu'il peut tourner jour et nuit pour un coût de quelques euros par an en électricité.

Concernant l'installation du système d'exploitation, il est fortement conseillé d'utiliser un noyaux Linux avec une distribution compatible telles que Raspbian, Debian, Ubuntu MATE, etc.

Après l'installation, nous avons connecté le Raspberry Pi au routeur afin de connaître son adresse IP par défaut afin de pouvoir modifier celle-ci à l'aide du fichier */boot/cmdline.txt*. De même, nous avons modifié son nom dans le but de simplifier l'identification de celui-ci lors de son utilisation ultérieure.

1. `https://www.raspberrypi.org/`

1.5 Sense HAT

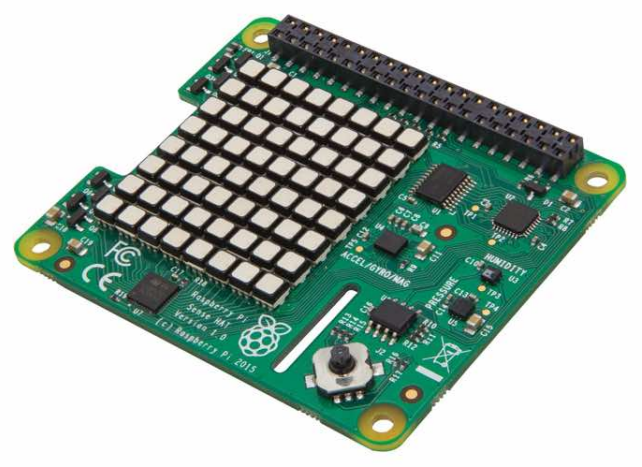


FIGURE 5 – Sense HAT

Le Sense HAT [2] est une carte additionnelle qui dispose d'une matrice LED 8x8, d'un joystick à cinq boutons et comprend les capteurs suivants :

— Gyroscope

— Accéléromètre

— Magnétomètre

— Température

— Pression barométrique

— Humidité

Remarque : une librairie permettant d'interagir avec les différents capteurs est disponible en Python.

Pour l'anecdote, le Sense HAT a été spécialement conçu pour la mission Astro Pi [3] lancé à la station spatiale internationale en décembre 2015.

2. `https://www.raspberrypi.org/products/sense-hat/`
3. `https://astro-pi.org/`

2 Choix

2.1 Langage

Pour ce projet, notre choix s'est naturellement porté vers le Python. En effet, en plus d'être le langage de prédilection sur le Raspberry Pi, il dispose d'une documentation et d'une vaste communauté.
En outre, ce langage apporte une facilité pour l'interaction avec les GPIO [4] de la Raspberry Pi et la librairie du Sense HAT.

2.2 Raspbian

Pour notre utilisation, Raspbian, étant le choix suggéré par le constructeur, a été choisi.

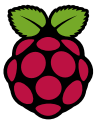


FIGURE 6 – Logo Raspbian

2.3 Outils de développement

GNU Emacs et Atom ont été les seuls éditeurs de texte utilisés comme outils de développement pour leur simplicité ainsi que pour notre familiarité avec ceux-ci.

FIGURE 7 – Logo Emacs

FIGURE 8 – Logo Atom

De plus, nous avons utilisé GitHub qui est un service en ligne permettant d'héberger notre projet et de ce fait, synchroniser notre travail.

FIGURE 9 – Logo GitHub

4. General-purpose input/output

3 Algorithmes de programmation

3.1 Organigramme du Pong

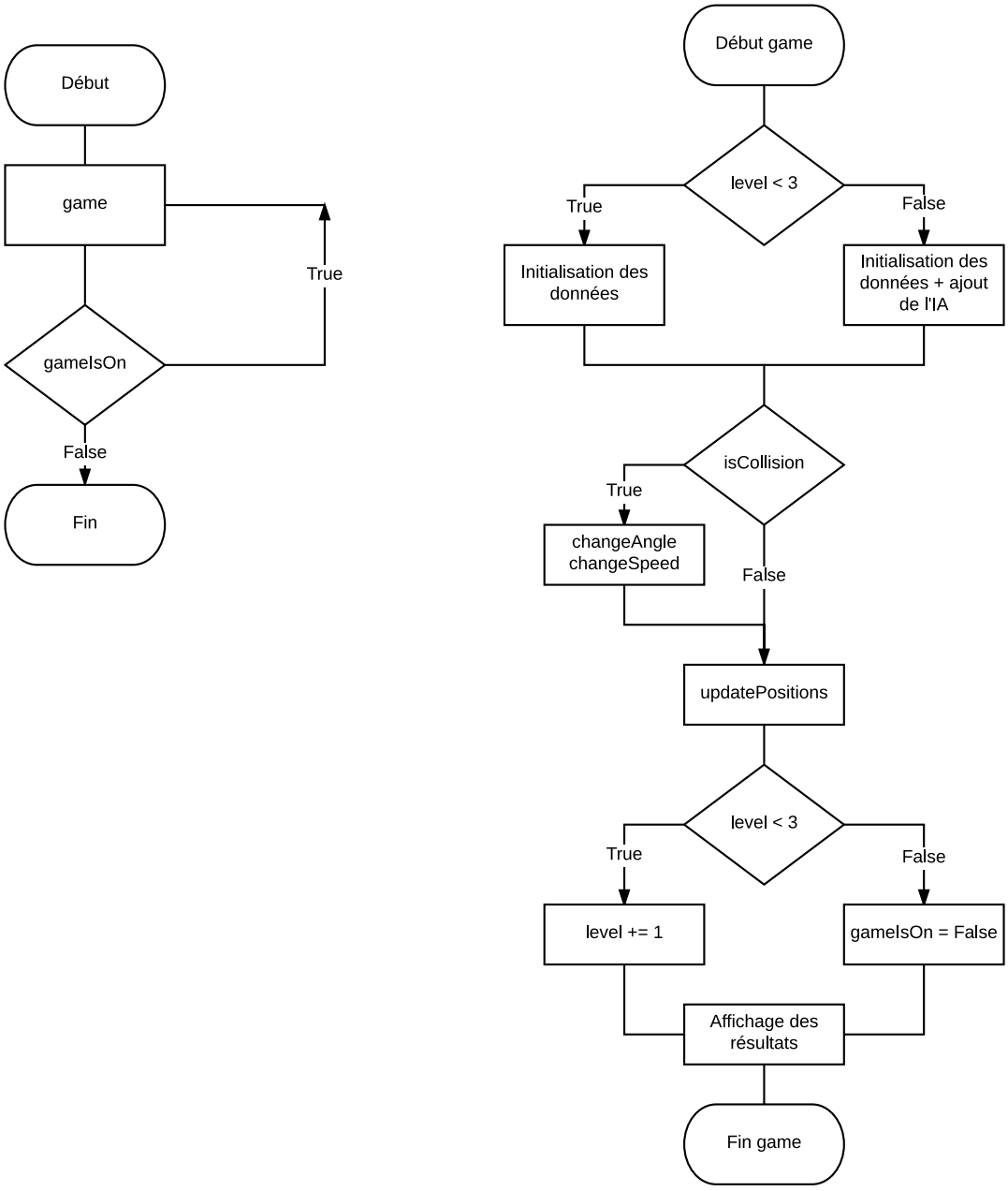


FIGURE 10 – Organigramme du PyPong

3.2 Collisions en détails

Dans ce point, il sera question de détailler le code de détection des différentes collisions du Pong. Liée à cela, la gestion des angles sera aussi explicitée.

Pour débuter, nous avons choisi - *au début du projet* - de définir le référentiel d'après la balle. Pour elle, l'axe des abscisses est l'axe sur lequel les raquettes se déplacent.

Collision avec les murs

```
def isWallCollision(self):
    if self.level < 3:
        if self.ball.x == 0 or self.ball.x == self.row - 1:
            #Roof
            self.ball.dx = -self.ball.dx
            self.result += 3
            self.ball.speed -= 0.1
        if self.ball.y == 0:
            #Left side
            self.ball.dy = -self.ball.dy
            self.result += 3
            self.ball.speed -= 0.1
    else:
        if self.ball.x == 0 or self.ball.x == self.row - 1:
            #Roof
            self.ball.dx = -self.ball.dx
            self.ball.speed -= 0.1
```

Dans ce cas, deux cas sont possibles : soit on est dans un des deux premiers niveaux et on joue seul contre le mur de gauche, soit on joue contre l'intelligence artificielle.

— Prenons le premier cas, la balle peut donc rebondir sur trois surfaces : celle du haut, celle du bas et celle de la gauche.

— Prenons le second cas où la balle ne peut rebondir que sur les bords supérieurs et inférieurs.

La collision avec le haut ou le bas se fait respectivement quand la position **x** est soit nulle, soit la dernière position possible dans les limites de la grille.
La collision avec le mur gauche se fait de manière similaire, mais seulement lorsque la position **y** est nulle que la balle entre en collision avec le mur.

Lors d'une collision, la vitesse et le résultat sont augmentés, et la direction **x** ou **y** est inversée.
Dans le cas où la balle va dans un coin gauche, elle subit les deux collisions : le joueur gagne le double de points et la vitesse est encore plus rapide.

Collision avec les raquettes

```
def isPaddleCollision(self):
    if self.level < 3:
        if self.ball.y == self.column - 2 and
        self.ball.x == self.paddle.x:
            self.ball.dy = -self.ball.dy
            self.result += 2
            self.ball.speed -= 0.15
    else:
        if (self.ball.y == 1 and
        self.ball.x == self.paddleLeft.x) or
        (self.ball.y == self.column - 2 and
        self.ball.x == self.paddleRight.x):
            self.ball.dy = -self.ball.dy
            self.ball.speed -= 0.15
```

Deux cas sont à nouveaux possibles : si le joueur se retouve contre l'intelligence artificielle, alors il y a deux raquettes en jeu. Sinon, la balle ne peut rebondir que sur la raquette de droite.

— Dans le cas sans l'intelligence artificielle, il suffit que la balle soit aux coordonnées de la raquette pour qu'il y ait une collision.
Lors de la collision, le résultat est augmenté de 2.

— Dans l'autre cas, la condition change légèrement : la balle doit toujours se trouver dans la colonne de la raquette *(la deuxième à gauche ou l'avant-dernière à droite)*, et à la ligne de la bonne raquette *(respectivement la gauche ou la droite)*.

S'il y a une collision avec une raquette, seule la direction **y** est inversée dans le but de garder des angles à 45°, et l'accélération est légèrement supérieure à celle observée lors de la collision avec un mur.

Sortie des limites

```
def isOut(self):
    if self.level < 3:
        return self.ball.y == self.column - 1
    return self.ball.y == 0 or self.ball.y == self.column - 1
```

Cette méthode est très simple.
Comme pour les autres collisions, il y aura le cas du niveau 3, et celui des niveaux inférieurs.

— Pour le niveau 3, la balle sort de la grille de jeu si elle touche le bord gauche ou le bord droit *(soit la première colonne, soit la dernière)*.

— Pour les autres niveaux, la balle pouvant, *(devant)*, rebondir sur le bord de gauche, seule une collision sur le bord droit est vue comme une sortie hors de la grille.

4 Problèmes rencontrés

4.1 Joystick

La librairie de python comportait une erreur dans la prise en charge du joystick [5]. Il a fallu supprimer les fichiers d'extension « *.py* » relatifs au Sense HAT et télécharger les fichiers mis à jour.

Pour terminer, il était nécessaire de supprimer un point mal placé dans l'une des importations de librairies.

4.2 Matrice

Étant donné la taille du Raspberry Pi, la matrice du Sense HAT s'est avérée fort petite pour l'implémentation du jeu de Pong traditionnel.

De ce fait, il y avait manque d'espace pour la taille des raquettes que nous avons été obligé de limiter à une LED, tout comme pour la balle, ce qui augmente la difficulté de prise en main du jeu.

De plus, les angles ont aussi été atteints par cette limitation matricielle, nous empêchant d'utiliser d'autres angles que des multiples de 45.

4.3 Gyroscope

Dans le cadre du projet, il nous a été demandé d'ajouter, comme fonctionnalité supplémentaire, une version du jeu se basant sur l'utilisation du gyroscope incorporé au Sense HAT.

Suite à un manque d'ergonomie lié à la manipulation du joystick et de l'inclinaison du gyroscope, nous avons décidé d'implémenter une intelligence artificielle (IA) comme autre fonctionnalité.

5. https ://www.element14.com/

5 Possibilités futures

Le code étant implémenté de manière soignée et en orienté objet, celui-ci offre beaucoup plus de possibilités pour d'éventuelles évolutions.

Lors de la conception de la version préliminaire du jeu, nous avons prévu plusieurs évolutions possibles pour notre Pong dans le cas où nous aurions plus de temps.

En voici quelques unes :

— le mouvement des raquettes sur deux axes : en plus de se déplacer sur l'axe vertical, les raquettes se seraient déplacées à l'horizontal.

— la taille des raquettes : plus le niveau serait haut, plus la raquette rapetisserait.

— l'ajout d'obstacles sur le terrain : des murs seraient placés de manière aléatoire sur le terrain pour augmenter la difficulté du jeu.

— les angles : avec une taille de raquette plus importante, l'implémentation de différents angles (50°, 70°, etc.) rendrait le jeu plus intéressant.

— la difficulté de l'IA : une seconde IA plus efficace se replacerait au centre de son axe dès que la balle rebondirait sur sa raquette.

6 Conclusion

6.1 Générale

Ce projet nous a permis d'apprendre à utiliser le Raspberry Pi, le Sense HAT ainsi que les nouveautés de Python 3 que nous n'avions pas encore eu la chance d'essayer. En outre, nous avons pu mieux comprendre le comportement du Sense HAT tel que vu au cours.

Nous avons donc implémenté un jeu de Pong en Python pour le Sense HAT, en tirant profit du joystick qui était à disposition.
Le jeu nous semblant « *fade* », il nous est venu à l'idée de lui rajouter un mode contre une intelligence artificielle.

Grâce à cela, nos connaissances en Python et en algorithmique ont été renforcées.
La prise en charge de collisions a été un challenge et un plus pour notre expérience, car totalement inédit dans nos précédents projets.

Ayant perdu 8 heures sur les 20 initialement prévues à cause du manque d'énoncé et de matériel lors des premières séances, nous avons dû nous dépasser pour terminer le projet dans les délais.
Cela nous a ainsi permis de nous améliorer dans notre gestion du temps.

6.2 Conclusions individuelles

> *Me concernant, j'ai déjà pu apprivoiser le langage Python à l'Université lors d'anciens projets, ainsi que pour des projets personnels. Néanmoins, ce projet m'a permis de manipuler un Raspberry Pi en fonction de l'algorithmique et prendre conscience de la complexité de celui-ci avec le hardware.*

Terencio Agozzino,

> *Grâce à ce projet, je me suis perfectionné en algorithmique par la prise en charge des angles liés aux collisions.*
> *Il y a un an, nous avions eu l'occasion de programmer un pic en C. Ce projet était ainsi une continuation m'ayant permis d'utiliser un Raspberry Pi, chose qui me tentait depuis plusieurs années.*

Alexandre Ducobu,

7 Code du projet

7.1 AI

```
class Ai(object):
    """
    Creates a new type of object: 'Ai'.
    """

    def __init__(self, paddle):
        """
        Constructs a new 'Ai' object.

        :param paddle: The ai paddle
        """
        self.paddle = paddle


    def distance(self, ball, paddle):
        """
        Calculates the distance between the paddle and the ball.

        :param ball:   The ball
        :param paddle: The ai paddle

        :return: Returns the distance between the paddle
                 and the ball.
        """
        return ball.y - paddle.y


    def goLeftUp(self, ball, paddle):
        """
        Verifies if the paddle can go up.

        :param ball:   The ball
        :param paddle: The ai paddle

        :return: return: Returns true if the paddle can go up,
                         false otherwise.
        """
        return ball.x < paddle.x
```

```
def goLeftDown(self, ball, paddle):
    """
    Verifies if the paddle can go down.

    :param ball:   The ball
    :param paddle: The ai paddle

    :return: return: Returns true if the paddle can go down,
                     false otherwise.
    """
    return ball.x > paddle.x


def debug(self, ball, paddle):
    """
    Useful debug method by displaying infos about the ai.

    :param ball:   The ball
    :param paddle: The ai paddle
    """
    print(' ' * 5 + '[ AI Mode ]' + ' ' * 5 + '\n')
    if (self.goLeftUp(ball, paddle)):
        print('Direction to take: UP')
    else:
        print('Direction to take: DOWN')
    print('Distance from the ball: ',
    str(self.distance(ball, paddle)) + '\n')

def play(self, ball, paddle):
    """
    Makes the ai play the game.

    :param ball:   The ball
    :param paddle: The ai paddle
    """
    if self.paddle.ID == 'Left' and ball.y <= 3:
        # self.debug(ball, paddle)
        if self.goLeftUp(ball, paddle):
            paddle.x = paddle.x - 1
        elif self.goLeftDown(ball, paddle):
            paddle.x = paddle.x + 1
    elif self.paddle.ID == 'Right' and ball.y >= 3:
        if self.goLeftUp(ball, paddle):
            paddle.x = paddle.x - 1
        elif self.goLeftDown(ball, paddle):
            paddle.x = paddle.x + 1
```

7.2 Ball

```
from component import *
from random import randrange

class Ball(Component):
    """
    Creates a new type of object: 'Ball'.
    """

    SYMBOL = 'o'
    NAME   = 'Ball'

    def __init__(self, x, y, side, speed, color):
        """
        Constructs a new 'Ball' object.

        :param x:     X-Axis of the ball
        :param y:     Y-Axis of the ball
        :param side:  Size of the ball
        :param speed: Speed of the ball
        :param color: Color of the ball
        """
        super(self.__class__, self).__init__(x, y, self.SYMBOL, color)
        self.side         = side
        self.dx           = 0
        self.dy           = 0
        t                 = [-135, 135]
        self.angle        = t[randrange(2)]
        self.InitialSpeed = speed
        self.speed        = speed
        self.setDirection()


    def __str__(self):
        """
        Displays the ball.
        """
        return str(self.color + self.SYMBOL + self.DEFAULT_COLOR)


    def __repr__(self):
        """
        Represents the ball.
        """
        return str(self.color + self.SYMBOL + self.DEFAULT_COLOR)
```

```
def setDirection(self):
    """
    Sets the direction the ball according to an angle.

      dx: vertical   (-1: up; 0: none; 1: down)
      dy: horizontal (-1: left; 0: none; 1: right)
    """
    if self.angle == -135:
        self.dx = 1
        self.dy = -1
    elif self.angle == -45:
        self.dx = 1
        self.dy = 1
    elif self.angle == 45:
        self.dx = -1
        self.dy = 1
    else:
        self.dx = -1
        self.dy = -1


def move(self):
    """
    Moves the ball according to the calculated direction.
    """
    self.x += self.dx
    self.y += self.dy
```

7.3 Component

```
from abc import ABC, abstractmethod

class Component(ABC):
    """
    Creates a new type of object: 'Component'.
    """

    DEFAULT_COLOR = '\033[0;0m'

    def __init__(self, x, y, symbol, color):
        """
        Constructs a new 'Component' object.

        :param x:      X-Axis of the component
        :param y:      Y-Axis of the component
        :param symbol: Symbol of the component
        :param color:  Color of the component
        """
        self.x      = x
        self.y      = y
        self.symbol = symbol
        self.color  = color


    def __str__(self):
        """
        Displays the component.
        """
        return str(self.color + self.SYMBOL + self.DEFAULT_COLOR)


    def __repr__(self):
        """
        Represents the component.
        """
        return str(self.color + self.SYMBOL + self.DEFAULT_COLOR)


    @abstractmethod
    def move(self):
        """
        Moves the component according to the X-Axis and Y-Axis.
        """
        pass
```

7.4 Game

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
sys.path.insert(0, '/usr/lib/python2.7/dist-packages/sense_hat')
from sense_hat import SenseHat

from ball import *
from paddle import *
from ground import *
from ai import *

import time
import os

# Copyright (C) 2017 Terencio Agozzino <terencio.agozzino@gmail.com>
#                    Alexandre Ducobu  <alexandre.ducobu@yahoo.be>
#
# PyPong is free software; you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as
# published by the Free Software Foundation, either version 3 of
# the License, or (at your option) any later version.
#
# PyPong is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU Lesser General Public License for more details.
#
# You should have received a copy of the GNU Lesser General Public
# License along with this program.
# If not, see <http://www.gnu.org/licenses/>.

YELLOW = "\033[1;33m"
sense = SenseHat()
sense.low_light = True
```

```
def displayInfo(fps, timeStart, timeElapsed):
    """
    Useful to get information about the game.

    :param fps:         Frames per second of the game
    :param timeStart:   Time start for the processus.
    :param timeElapsed: Time elapsed for the processus.
    """
    print(' ' * 5 + '[ Game Info ]' + ' ' * 5 + '\n' * 2
          + 'FPS: ' + str(fps) + '\n'
          + 'Time Elapsed: ' + str(timeElapsed) + '\n'
          + 'Time Start: ' + str(timeStart) + '\n')


def render(ground):
    """
    Renders the game.

    :param ground: Ground of the game
    """

    game = ground.update()
    printArray(ground.array)
    time.sleep(ground.ball.speed)
    ground.ball.speed = ground.ball.InitialSpeed
    os.system('clear')
    return game


def printArray(array):
    """
    Displays an array.

    :param array: Array
    """
    for j in range(len(array)):
       for i in range(len(array)):
          if array[i][j] == 0:
             sense.set_pixel(j, i, [0, 0, 0])
          elif isinstance(array[i][j], Ball):
             sense.set_pixel(j, i, [255,255,255])
          else:
             sense.set_pixel(j, i, [173, 0, 0])
```

```
def start():
    """
    Starts the game.
    """

    level = 1

    frames = []
    timeStart = time.process_time()

    game = True

    sense.show_message("PyPong")
    time.sleep(0.2)
    sense.show_message("3")
    time.sleep(0.1)
    sense.show_message("2")
    time.sleep(0.1)
    sense.show_message("1")
    time.sleep(0.1)
    sense.show_message("Go!")

    player1 = 0
    player2 = 0

    speed = 0.6
    games = 0

    while (player1 != 3 and player2 != 3):
        games += 1
        ball = Ball(4, 4, 1, speed, YELLOW)
        ground = Ground(8, 8, ball, level)
        sense.stick.direction_up = ground.pushedUp
        sense.stick.direction_down = ground.pushedDown
        gameIsOn = True
        while (gameIsOn):
            gameIsOn, result = render(ground)
            if gameIsOn:
                timeEnd = time.process_time()
                timeElapsed = timeEnd - timeStart

                timeStart = timeEnd

                frames.append(timeElapsed)
                frames = frames[-20:]
                fps = len(frames) / sum(frames)
```

```
            if level < 3:
                sense.show_message(str(result))
                sense.show_message('points')
                if games == 3:
                    level += 1
                    speed -= 0.01
                    games = 0
                    sense.show_message('Level ' + str(level) + '!')
            else:
                if result == 1:
                    player1 = player1 + 1
                else:
                    player2 = player2 + 1
                sense.show_message(str(player1) + ':' + str(player2))
            speed -= 0.02

            time.sleep(1)
        sense.show_message("End")


if __name__ == '__main__':
    start()
```

7.5 Ground

```
from ball import *
from paddle import *
from ai import *

import time

class Ground(object):
    """
    Creates a new type of object: 'Ground'.
    """

    RED     = "\033[1;31m"
    BLUE    = "\033[1;34m"
    YELLOW  = "\033[1;33m"
    CYAN    = "\033[1;36m"
    GREEN   = "\033[0;32m"
    DEFAULT = "\033[0;0m"
    BOLD    = "\033[;1m"
    stickMove = 0


    def __init__(self, row, column, ball, level):
        """
        Constructs a new 'Ground' object.

        :param row:    X-Axis length of the ground
        :param column: Y-Axis length of the ground
        :param ball:   Ball of the ground
        :param level:  Level of the ground
        """
        self.row         = row
        self.column      = column

        self.ball        = ball

        self.level       = level
        self.result      = 0

        if self.level < 3:
            self.paddle      = Paddle((self.row // 2) + 1,
                self.column-2, 'Right', 2, self.RED)
            self.components  = [self.ball, self.paddle]
            self.positions   = [ball.y, ball.x, self.paddle.y,
                self.paddle.x]
```

```
        else:
            self.paddleLeft  = Paddle(self.row // 2, 1, 'Left',
                2, self.RED)
            self.paddleRight = Paddle((self.row // 2) + 1,
                self.column-2, 'Right', 2, self.CYAN)
            self.components  = [self.ball, self.paddleLeft,
                self.paddleRight]
            self.positions   = [ball.y, ball.x, self.paddleLeft.y,
                self.paddleLeft.x, self.paddleRight.x,
                self.paddleRight.y]
            self.ai          = Ai(self.paddleLeft)

        self.array       = self.generateGround(row, column)

        self.initComponents(self.components)


    def __str__(self):
        """
        Displays the ground.
        """
        return str(self.array[i])


    def cleanPositions(self):
        """
        Removes the old positions of the components.
        """
        self.array[self.positions[0]][self.positions[1]] = 0
        self.array[self.positions[2]][self.positions[3]] = 0
        if self.level == 3:
            self.array[self.positions[4]][self.positions[5]] = 0

    def debug(self, components):
        """
        Useful debug method by displaying X-Axis and Y-Axis
        of components.

        :param components: List of components
        """
        print(' ' * 5 + '[ Debug Mode ]' + ' ' * 5 + '\n')
        for component in components:
            print(component.NAME + "(x=" + str(component.x)
                                 + ", y=" + str(component.y) + ")")
        print('\n')
```

```
def generateGround(self, row, column):
    """
    Generates the ground.

    :param row:    Number of rows for the ground
    :param column: Number of columns for the ground
    """
    return [[0 for i in range(row)] for j in range(column)]


def initComponents(self, components):
    """
    Initializes the components in the array.

    :param components: List of components
    """
    for component in components:
        self.array[component.x][component.y] = component


def isWinner(self):
    """
    Returns the result of the current game.

    :return: Returns the score if the level is lower than 3,
             1 if the first player wins, 2 for the second.
    """
    if self.level < 3:
        return self.result
    if self.ball.y < 0:
        return 2
    elif self.ball.y > 7:
        return 1


def isPaddleCollision(self):
    """
    If there is a collision with a paddle,
    the direction of the ball changes et the score is upgraded.
    """
    if self.level < 3:
        if self.ball.y == self.column - 2 and
            self.ball.x == self.paddle.x:
            self.ball.dy = -self.ball.dy
```

```
                self.result += 2
                self.ball.speed -= 0.15
        else:
            if (self.ball.y == 1 and
              self.ball.x == self.paddleLeft.x) or
              (self.ball.y == self.column - 2 and
              self.ball.x == self.paddleRight.x):
                self.ball.dy = -self.ball.dy
                self.ball.speed -= 0.15


    def isOut(self):
        """
        Detects if the ball is out of bounds.

        :return: Returns true if the ball is out of bounds,
                         false otherwise.
        """
        if self.level < 3:
            return self.ball.y == self.column - 1
        return self.ball.y == 0 or self.ball.y == self.column - 1


    def isWallCollision(self):
        """
        Detects if there is a collision with the ball
        and the bottom or top of the wall or with the
        right side of the ground.
        """
        if self.level < 3:
            if self.ball.x == 0 or self.ball.x == self.row - 1:
                #Roof
                self.ball.dx = -self.ball.dx
                self.result += 3
                self.ball.speed -= 0.1
            if self.ball.y == 0:
                #Left side
                self.ball.dy = -self.ball.dy
                self.result += 3
                self.ball.speed -= 0.1
        else:
            if self.ball.x == 0 or self.ball.x == self.row - 1:
                #Roof
                self.ball.dx = -self.ball.dx
                self.ball.speed -= 0.1
```

```
def pushedUp(self):
    """
    If the joystick of the Sense HAT is pushed up,
    the paddle is moved upwards.
    """
    self.stickMove = -1


def pushedDown(self):
    """
    If the joystick of the Sense HAT is pushed down,
    the paddle is moved downwards.
    """
    self.stickMove = 1


def movePaddle(self, move):
    """
    Verifies if the paddle can move.
    If it can, it'll from the direction given by the Sense HAT.
    """
    if self.level < 3:
        if not ((self.paddle.x == 0 and self.stickMove == -1)
          or (self.paddle.x == self.row-1
          and self.stickMove == 1)):
            self.paddle.x += move
    else:
        if not ((self.paddleRight.x == 0 and
          self.stickMove == -1) or
          (self.paddleRight.x == self.row-1 and
          self.stickMove == 1)):
            self.paddleRight.x += move


def update(self):
    """
    Updates the game.
    """
    self.cleanPositions()
    out = True

    if self.isOut():
        out = False

    if self.level < 3:
```

```
            self.movePaddle(self.stickMove)

            if out:
                self.isWallCollision()
                self.isPaddleCollision()

            self.positions[0] = self.ball.x
            self.positions[1] = self.ball.y

            self.positions[2] = self.paddle.x
            self.positions[3] = self.paddle.y

            self.array[self.positions[2]][self.positions[3]] =
                self.paddle
        else:
            self.movePaddle(self.stickMove)

            self.isWallCollision()
            self.isPaddleCollision()

            self.positions[0] = self.ball.x
            self.positions[1] = self.ball.y

            self.positions[2] = self.paddleLeft.x
            self.positions[3] = self.paddleLeft.y
            self.positions[4] = self.paddleRight.x
            self.positions[5] = self.paddleRight.y

            self.array[self.positions[2]][self.positions[3]] =
                self.paddleLeft
            self.array[self.positions[4]][self.positions[5]] =
                self.paddleRight

        self.array[self.positions[0]][self.positions[1]] = self.ball

        self.ball.move()
        if self.level == 3:
            self.ai.play(self.ball, self.paddleLeft)
            winner = self.isWinner()
        else:
            winner = self.isWinner()

        self.stickMove = 0

        return out, winner
```

7.6 Paddle

```
from component import *

class Paddle(Component):
    """
    Creates a new type of object: 'Paddle'.
    """

    SYMBOL = '|'
    NAME   = 'Paddle'


    def __init__(self, x, y, ID, size, color):
        """
        Constructs a new 'Paddle' object.

        :param x:     X-Axis of the paddle
        :param y:     Y-Axis of the paddle
        :param ID:    ID of the paddle
        :param size:  Size of the paddle
        :param color: Color of the paddle
        """
        super(self.__class__, self).__init__(x, y, self.SYMBOL, color)
        self.ID        = ID
        self.size      = size
        self.direction = 1 #-1: up; 1: down; 0 fixe

    def __str__(self):
        """
        Displays the paddle.
        """
        return str(self.color + self.SYMBOL + self.DEFAULT_COLOR)

    def __repr__(self):
        """
        Represents the paddle.
        """
        return str(self.color + self.SYMBOL + self.DEFAULT_COLOR)

    def move(self):
        """
        Moves the paddle.
        """
        self.y += self.direction
```

Références

[1] Communauté de la mission Astro Pi. Astro Pi Mission, *informatique*, Site, [en ligne]. https ://astro-pi.org, (consulté le 07 Mars 2017).

[2] Communauté de Wikipédia. Pong, *informatique*, Site, [en ligne]. https ://fr.wikipedia.org/wiki/Pong, (consulté le 07 Mars 2017).

[3] LaTeX. A document preparation system. (consulté le 07 Mars 2017).

[4] The Raspberry Pi Foundation. Python module to control the Raspberry Pi Sense HAT, *informatique*, Site, [en ligne]. https ://pythonhosted.org/sense-hat/, (consulté le 07 Avril 2017).

[5] The Raspberry Pi Foundation. SENSE HAT, *informatique*, Site, [en ligne]. https ://www.raspberrypi.org/products/sense-hat/, (consulté le 07 Avril 2017).

[6] The Raspberry Pi Foundation. DOCUMENTATION, *informatique*, Site, [en ligne]. https ://www.raspberrypi.org, (consulté le 07 Mars 2017).

[7] Tozer, L. Raspberry Pi Sense Hat - Enabling The Joystick, *informatique*, Site, [en ligne]. https ://www.element14.com, (consulté le 07 Avril 2017).