IMPLEMENTACIÓN EN PYTHON DEL JUEGO AHORCADO

(Septiembre de 2020)

Ivan Fernando Mujica Mamani  
Universidad Catolica Boliviana San Pablo

**Resumen**—En años recientes el campo de la ciencia de datos ha tomado especial importancia en la toma de decisiones de las empresas e instituciones, una parte vital de este campo es el dominio de un lenguaje con una amplia implementación de librerías y paquetes para el manejo e interpretación de datos; Python es uno de los lenguajes más populares para la aplicación de ciencia de datos, en parte gracias a que tiene una de las curvas de aprendizaje menos pronunciadas. En el presente artículo implementaremos un algoritmo para el popular juego *Ahorcado* el cual básicamente se trata de adivinar una determinada palabra letra por letra.

**Índice de términos**—Ahorcado, Python, Ciencia de Datos

—————————— ◆ ——————————

# 1 Introduccion

En el presente artículo explicamos cómo se implementa el juego Ahorcado en el lenguaje de programación Python en su versión 3.8, para tal efecto no se usó ninguna librería perteneciente al core del lenguaje. Debido a que este lenguaje implementa tipado dinámico y programación funcional es mucho más sencillo y la curva de aprendizaje es menos pronunciada en relación a otros lenguajes como Java o C++[1]. Es necesario contar con un ambiente de desarrollo con Python y algún editor de código.

Se necesita un programa que permita adivinar al jugador una palabra letra por letra, con las siguientes restricciones semánticas:

1. Se tiene un número máximo de intentos
2. El jugador debe adivinar letra por letra, si existe una letra más de una vez; se descubren todas.
3. Si el jugador no adivina una letra, el contador de errores se incrementa.
4. El juego termina cuando el total de errores es igual al número de intentos máximo para el juego.
5. Si el jugador ingresa una letra adivinada anteriormente, no se contabiliza como error o acierto.
6. Adicionalmente se puede ilustrar con un gráfico (ASCII o similar ) de una persona siendo colgada progresivamente en cada error.

# 2 Resolución **del problema**

## 2.1 Definición **de funciones y variables globales**

El punto de entrada del programa es la función llamada *main* donde se tiene un prompt de entrada para preguntar al jugador si quiere proceder con el juego, la función *play* es el que ejecuta el juego; por lo tanto y a su vez, recibe como único argumento la palabra que se tiene adivinar.

def main():

yesNo = input('Wanna start the game?[Yes/No]: ')

if yesNo.lower() in ['yes', 'no']:

if yesNo.lower() == 'yes' :

play('paciencia') # set dynamic spanish word generator

if yesNo.lower() == 'no':

print('Ok. Sorry for asking')

else :

print('Please answer Yes or No')

main()

main()

## 2.2 Declaración de variables en la función *play*

La variable *hidden* de tipo array nos sirve para volcar parcialmente las letras adivinadas de la palabra original, la variable *word* se asigna a sí misma como un array de letras, la variable *attempts* contabiliza cuando errores comete el jugar en cada iteración, la variable *max\_attempts* sirve para controlar el límite de errores que puede tener el jugador.

hidden = []

attempts = 0

max\_attempts = 4

word = list (word)

El array *hidden* se inicia llenado con un carácter especial diferencial en este caso “\_”.

for char in word :

hidden.append('\_')

Por último usamos la variable *isGameOver* de tipo bandera que le permite iterar al flujo de control hasta que haya un resultado del juego.

## 2.3 Iteración **principal**

La iteración principal está compuesta por 4 bloques. La primera es un mensaje al jugador con el número, la segunda es la ilustración en ASCII de una persona colgada y solicita al jugador ingresar una letra, el tercero es una evaluación contra la variable word, si la letra ingresada está dentro de la palabra se hace un movimiento de la letra entre del array word al hidden; si la letra ingresada no existe en la palabra se incrementa en 1 la variable *attempts*, el bloque final de este flujo de control pregunta si el array word ya no tiene letras, asumiendo que todas están en la variable *hidden*, entonces el el flujo de control se termina y devolvemos un mensaje de victoria al jugador, caso contrario preguntamos si la variable *attempts* es igual o mayor a *max\_attempts* para romper el flujo e indicar al jugador que perdió el juego.

while not isGameOver:

# first part

print(f'You still have {max\_attempts - attempts} attemps remaining')

hiddenString = ''.join(hidden)

print(f'The current word is : {hiddenString}')

# second part

print(' \_\_\_\_\_\_\_\_\_')

print(' | |')

print(' | ' + (' O' if attempts > 0 else ''))

print(' | ' + ('/ \\' if attempts > 1 else ''))

print(' | ' + (' |' if attempts > 2 else ''))

print(' | ' + ('/ \\' if attempts > 3 else ''))

print('-------- ')

letterGuessed = input('Enter a letter: ')

print('\n\n\n')

# third part

if letterGuessed in word:

print(f'Good, the letter {letterGuessed} is in the word.')

for i in range(len(word)):

if word[i] == letterGuessed:

hidden[i] = word[i]

word[i] = '\_'

else :

if letterGuessed not in hidden:

attempts += 1

print(f'Wrong! The letter {letterGuessed} is not in the word.')

else:

print(f'You have already used letter {letterGuessed}')

# print(word, hidden)

# fourth part

# asking if there is any letter unguessed

if all('\_' == character for character in word):

print (f'Contratulations, YOU WIN!!')

isGameOver = True

# asking if attemps are equal to max\_attempts

if attempts >= max\_attempts :

print(f'So sorry, YOU LOOSE!!')

isGameOver = True

print(' \_\_\_\_\_\_\_\_\_')

print(' | |')

print(' | O')

print(' | / \\')

print(' | |')

print(' | / \\')

print('----------- ')

Adicionalmente se hace una llamada más a la función main principal para reiniciar el juego.

# Conclusion

Por lo expuesto en el presente artículo podemos apreciar la flexibilidad del lenguaje de programación en cuanto a la mensajería con el Sistema Operativo se refiere, no se necesito usar ninguna librería del core del lenguaje. La implementacion completa del juego se encuentra en <https://github.com/ifmm87/introduction-data-science/blob/master/exercises-1109/hangman.py>

**References**

1. M. Summerfield, *Programming in Python 3 A complete introduction to the Python language, 2nd ed.* Boston, MA.: Pearson Education, pp. 2-3, 2010.