Comisión:03

Cátedra: Fernando Velcic Francisco Orozco De La Hoz
Alumnos :Mansilla Miguel , Lizarraga Jorge
Institución : Universidad Nacional De General Sarmiento

Introducción a la programación

Juego tiene la palabra

Reglas:

El juego consiste en que el usuario ingrese palabras de al menos 3 letras. Puedes repetir las letras, pero siempre incluyendo la letra principal. No se admiten plurales y formas verbales conjugadas (solo infinitivos). Puntuación: las palabras de 3 letras dan 1 punto y las de 4 letras, 2 puntos. A partir de 5 letras, se obtendrá tantos puntos como letras tenga la palabra. Los heptacracks (palabras de 7 letras) valen 10 puntos y cada error resta 1 punto.

Intervención:

Decidimos crear una pantalla principal que deje al usuario seleccionar la dificultad, dependiendo el nivel que se elija, cambiará el fondo, la música y el tiempo de jugabilidad.

- El modo fácil tendrá un tiempo de juego de 60 segundos con una imagen de naturaleza y una música agradable.

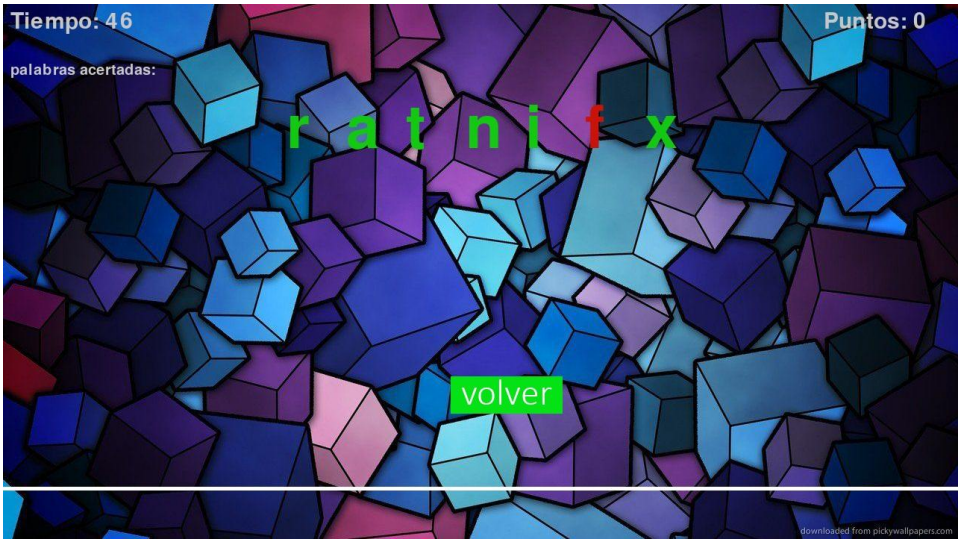
- El modo medio tendrá un tiempo de juego de 50 segundos con imagen y música agradable

- el modo difícil tendrá un tiempo de juego de 40 segundos con imagen y música agresiva

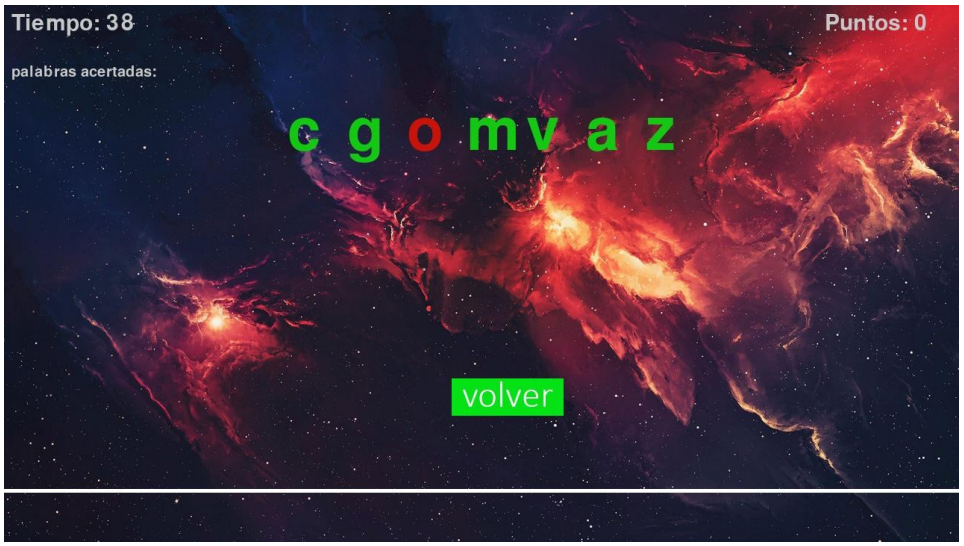
Interfaz Gráfica:

Pantalla principal de selección de dificultad.

Juego en dificultad fácil.



Juego en dificultad medio.



Juego en dificultad difícil.

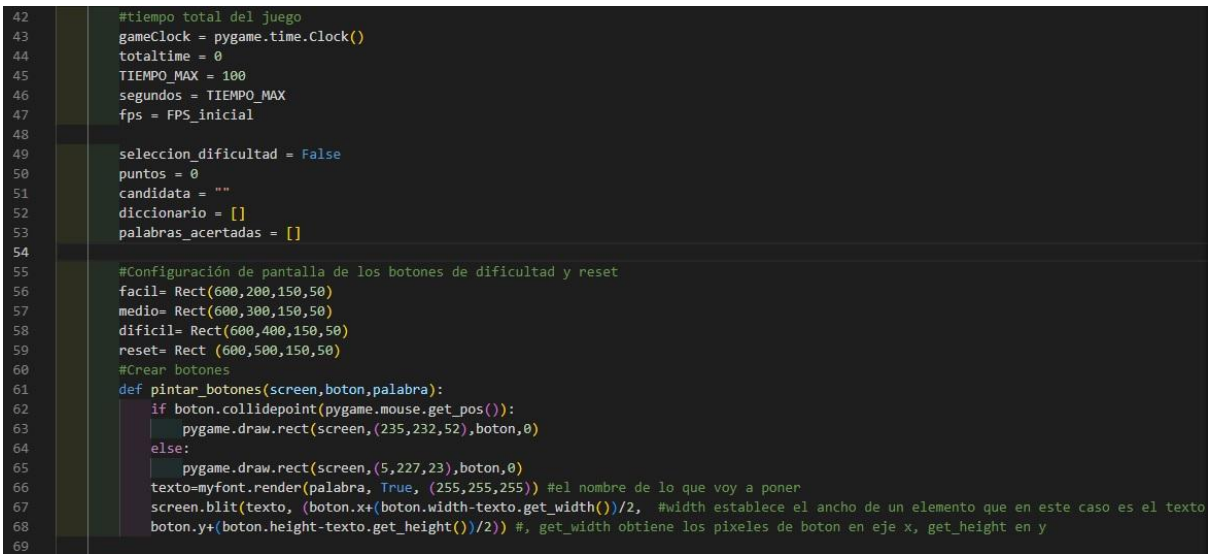
A continuación daremos una explicación de cada una de las funcionalidades y modificaciones que hemos hecho en el código.

principal.py

De la línea 21 a la 35 nos encargamos de crear todas las variables y llamadas que tengan que ver con el fondo, los sonidos y la música.

```
20      #fondo
21      fondo=pygame.image.load("assets/imagenes/Fondo.jpg")
22      imagen_fondo = fondo
23      Verano=pygame.image.load("assets/imagenes/Verano.jpg")
24      Galaxia=pygame.image.load("assets/imagenes/Galaxia.jpg")
25      Cuadrado=pygame.image.load("assets/imagenes/Cuadrado.jpg")
26
27      #sonidos
28      sonido_correcto = pygame.mixer.Sound("assets/sonidos/correct-ding.mp3")
29      sonido_error = pygame.mixer.Sound("assets/sonidos/Error.mp3")
30      #musica
31      Heroic_Age = pygame.mixer.Sound("assets/sonidos/Heroic_Age.mp3")
32      Infierno = pygame.mixer.Sound("assets/sonidos/Infierno.mp3")
33      The_Entertainer=pygame.mixer.Sound("assets/sonidos/The_Entertainer.mp3")
34      Magic=pygame.mixer.Sound("assets/sonidos/Magic.mp3")
35      Heroic_Age.play()
36
```

En la línea 45 decidimos declarar el tiempo_max ya que necesitábamos que sea una variable que pueda modificarse y desde el archivo de configuración no lo podemos hacer.
En la línea 49 creamos la variable de tipo bool selección_dificultad para luego ser usada en el while principal. Esta variable nos da la posibilidad de crear una pantalla principal que nos permite elegir la dificultad del juego.
De la línea 56 a la 68 le damos configuración de ubicación en pantalla y estilos a los botones de dificultad.



```
42      #tiempo total del juego
43      gameClock = pygame.time.Clock()
44      totaltime = 0
45      TIEMPO_MAX = 100
46      segundos = TIEMPO_MAX
47      fps = FPS_inicial
48
49      seleccion_dificultad = False
50      puntos = 0
51      candidata = ""
52      diccionario = []
53      palabras_acertadas = []
54
55      #Configuración de pantalla de los botones de dificultad y reset
56      facil= Rect(600,200,150,50)
57      medio= Rect(600,300,150,50)
58      dificil= Rect(600,400,150,50)
59      reset= Rect (600,500,150,50)
60      #Crear botones
61      def pintar_botones(screen,boton,palabra):
62          if boton.collidepoint(pygame.mouse.get_pos()):
63              pygame.draw.rect(screen,(235,232,52),boton,0)
64          else:
65              pygame.draw.rect(screen,(5,227,23),boton,0)
66          texto=myfont.render(palabra, True, (255,255,255)) #el nombre de lo que voy a poner
67          screen.blit(texto, (boton.x+(boton.width-texto.get_width())/2,  #width establece el ancho de un elemento que en este caso es el texto
68          boton.y+(boton.height-texto.get_height())/2)) #, get_width obtiene los pixeles de boton en eje x, get_height en y
69
```

A partir de la línea 86 entramos en el while principal que corre el bucle del juego, de la línea 88 a la 118 nos encargamos de hacer las validaciones correspondientes para que en primera instancia veamos los botones de dificultad. Como mencionamos anteriormente, la variable selección dificultad que se declara en False, nos da la posibilidad de comenzar el juego mostrando los botones, dependiendo de qué dificultad elijamos, la imagen de fondo, la música y el tiempo cambiaran, dando paso a cambiar la variable de seleccion_dificultad a True para que podamos continuar con la ejecución del resto del código.

En la línea 116 tenemos una validación que nos permite reiniciar el juego en caso de apretar el botón volver.

```
86      while segundos > fps/1000:
87
88          if seleccion_dificultad == False:
89              pintar_botones(screen,facil,"Facil")
90              pintar_botones(screen,medio,"Medio")
91              pintar_botones(screen,dificil,"Dificil")
92
93          if True:
94              fps = 30
95          #Buscar la tecla apretada del modulo de eventos de pygame
96          for e in pygame.event.get():
97              if e.type == MOUSEBUTTONDOWN and e.button==1:
98                  if facil.collidepoint(pygame.mouse.get_pos()):
99                      pygame.mixer.stop()
100                     imagen_fondo = Verano
101                     The_Entertainer.play()
102                     TIEMPO_MAX = 60 + pygame.time.get_ticks() / 1000
103                     seleccion_dificultad = True
104                 if medio.collidepoint(pygame.mouse.get_pos()):
105                     pygame.mixer.stop()
106                     imagen_fondo = Cuadrado
107                     Magic.play()
108                     TIEMPO_MAX = 50 + pygame.time.get_ticks() / 1000
109                     seleccion_dificultad = True
110                 if dificil.collidepoint(pygame.mouse.get_pos()):
111                     pygame.mixer.stop()
112                     imagen_fondo = Galaxia
113                     Infierno.play()
114                     TIEMPO_MAX = 40 + pygame.time.get_ticks() / 1000
115                     seleccion_dificultad = True
116                 if reset.collidepoint(pygame.mouse.get_pos()):
117                     pygame.mixer.stop()
118                     main()
```

Luego de apretar los botones de dificultad y de cambiar el valor de selección dificultad a True, nos da la posibilidad de continuar con la ejecución del código que nos permite escribir letras por teclado, acumular puntos, ejecutar los sonidos en caso de acierto o error, acumular las palabras acertadas en una lista,etc.

En la línea 148 ponemos la condición de que cuando los segundos sean menores a 0.5 , la música se pare, congele la ejecución del juego durante 10 segundos y vuelva a reiniciar el juego.

```
125             if seleccion_dificultad == True:
126                 #Ver si fue apretada alguna tecla
127                 if e.type == KEYDOWN:
128                     letra = dame_letra_apretada(e.key)
129                     candidata += letra   #va concatenando las letras que escribe
130                     if e.key == K_BACKSPACE:
131                         candidata = candidata[0:len(candidata)-1] #borra la ultima
132                     if e.key == K_RETURN:  #presionó enter
133                         puntos += procesar(letra_principal, letras_en_pantalla, candidata, diccionario , palabras_acertadas)
134                         #Verificamos el numero que retorna la función procesar y dependiendo de eso emitimos un sonido
135                         if (procesar(letra_principal, letras_en_pantalla , candidata, diccionario, palabras_acertadas) > 0):
136                             sonido_correcto.play()
137                         else:
138                             sonido_error.play()
139                         #Verificamos si la palabra candidata es valida, si es valida la guardamos en la lista de palabras_acertadas
140                         if (es_valida(letra_principal, letras_en_pantalla, candidata , diccionario, palabras_acertadas) == True):
141                             palabras_acertadas.append(candidata)
142                         candidata = ""
143         #Pintamos el boton si esta estamos dentro del juego
144         if seleccion_dificultad == True:
145             pintar_botones(screen, reset, "volver")
146         segundos = TIEMPO_MAX - pygame.time.get_ticks() / 1000
147         #verificamos que el tiempo sea menor a 0.5, frenamos la musica, congelamos la pantalla durante 10 segundos y reiniciamos el juego
148         if segundos < 0.5:
149             pygame.mixer.stop()
```

funcionesVacias.py

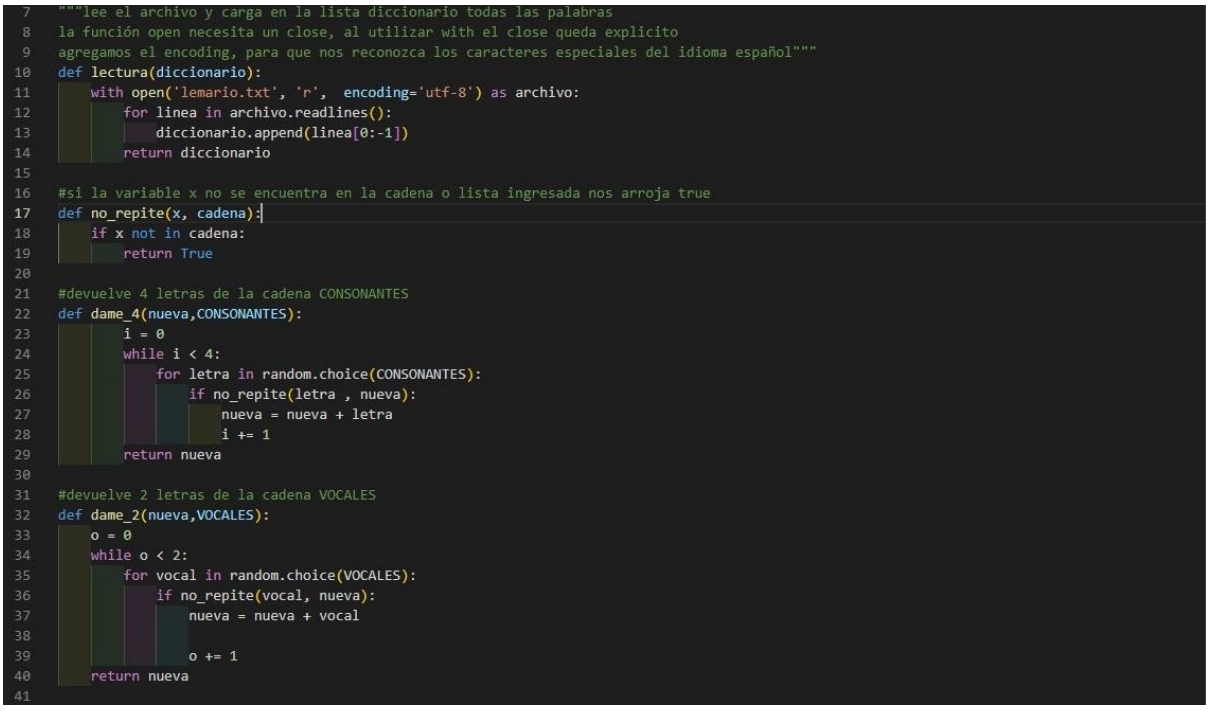
En este archivo se encuentran todas las funcionalidades que tiene el juego, a continuación daremos una breve explicación de cada una de ellas.

def lectura: recibe el parámetro diccionario, utilizamos la función with open para abrir el archivo temario.txt que contiene una lista de palabras y la guardamos con el nombre archivo. Luego recorremos todas las palabras del archivo con un for y las vamos guardando una a una a través de un append en la lista diccionario, luego retornamos la lista diccionario.

def no_repite: recibe x y una cadena como parámetro y hacemos una validación que dice que si el primer parámetro no se encuentra dentro de la cadena nos retorna un booleano de True.

def dame_4: Recibe el parámetro nueva, que es una cadena vacía, y CONSONANTES que es una variable constante que tiene todas las letras consonantes adentro, a través de un bucle while que nos permita realizar 4 iteraciones, haremos que un for recorra la cadena CONSONANTES de forma random, y las vaya asignando en cada vuelta dentro de letra. luego hacemos la condición de la función de no_repite() para que no se puedan repetir las letras que que se vayan guardando en la variable nueva, una vez que la variable nueva contenga 4 letras, la función retorna la variable nueva cargada con 4 letras

def dame_2: Recibe el parámetro nueva, que es una cadena vacía, y VOCALES que es una variable constante que tiene todas las letras vocales adentro, a través de un bucle while que nos permita realizar 2 iteraciones, haremos que un for recorra la cadena VOCALES de forma random, y las vaya asignando en cada vuelta dentro de vocal.
Luego hacemos la condición de la función de no_repite() para que no se puedan repetir las letras que se vayan guardando en la variable nueva, una vez que la variable nueva contenga 2 letras, la función retorna la variable nueva cargada con 2 letras.



```python
"""lee el archivo y carga en la lista diccionario todas las palabras
la función open necesita un close, al utilizar with el close queda explicito
agregamos el encoding, para que nos reconozca los caracteres especiales del idioma español"""
def lectura(diccionario):
    with open('lemario.txt', 'r',  encoding='utf-8') as archivo:
        for linea in archivo.readlines():
            diccionario.append(linea[0:-1])
        return diccionario

#si la variable x no se encuentra en la cadena o lista ingresada nos arroja true
def no_repite(x, cadena):
    if x not in cadena:
        return True

#devuelve 4 letras de la cadena CONSONANTES
def dame_4(nueva,CONSONANTES):
        i = 0
        while i < 4:
            for letra in random.choice(CONSONANTES):
                if no_repite(letra , nueva):
                    nueva = nueva + letra
                    i += 1
        return nueva

#devuelve 2 letras de la cadena VOCALES
def dame_2(nueva,VOCALES):
    o = 0
    while o < 2:
        for vocal in random.choice(VOCALES):
            if no_repite(vocal, nueva):
                nueva = nueva + vocal

                o += 1
    return nueva

```

def dame_1_dificil: Recibe el parámetro nueva, que es una cadena vacía, y C_DIFICIL que es una variable constante que tiene todas las letras difíciles (kxyz) adentro.
haremos que un for recorra la cadena C_DIFICIL de forma random, y las vaya asignando en cada vuelta dentro de letra_dificil.
Luego hacemos la condición de la función de no_repite() para que no se puedan repetir las letras que se vayan guardando en la variable nueva, una vez que la variable nueva tenga su letra, la función retorna la variable nueva cargada con una letra dificil.

def desordenar_cadena : Recibe el parámetro nueva , que contiene una cadena de 7 letras , y un parámetro de una cadena vacía llamada desordenada.
Creamos un while que itere 7 veces, luego agarramos la cadena nueva y mediante el for vamos recorriendo cada una de las letras y guardandolas en i, luego hacemos la condición de la función de no_repite() para que cada vez que de true nos guarde en desordenada 1 letra y nos sume 1 al iterador del while.
Luego retorna la cadena desordenada.
Esta función nos da la posibilidad de que las letras que aparezcan en la pantalla salgan sin el orden de ejecución de las funciones que guardan las letras en la variable nueva.

def dame_7_letras: Esta función se encarga de guardar en la variable nueva una cadena de 7 letras de las cuales 4 son consonantes, 2 son vocales y una es una letra difícil.
Nos encargamos de concatenar los valores que nos devuelven las funciones dame_4_letras, dame_2_letras y dame_1_dificil en la variable nueva para luego pasarsela por parámetro a la función desordenar_cadena y así obtener una cadena desordenada.

def dame_letra: Esta función recibe la variable letras_en_pantalla la cual contiene el retorno de la función dame_7_letras().
agregamos un for que recorra de manera random todas las letras y que elija una de las 7 y la retorne.

```python
#devuelve una letra de la cadena C_DIFICIL
def dame_1_dificil(nueva, C_DIFICIL):
    for letra_dificil in random.choice(C_DIFICIL):
        if no_repite(letra_dificil , nueva):
            nueva = nueva + letra_dificil

    return nueva
#desordena una cadena
def desordenar_cadena(nueva, desordenada):
    u = 0
    while u < 7:
        for i in random.choice(nueva):
            if no_repite(i , desordenada):
                desordenada = desordenada + i
                u += 1
    return desordenada

#Devuelve una cadena de 7 caracteres sin repetir con 2 o 3 VOCALES y a lo sumo
# con una consonante dificil (kxyz)
def dame_7_letras():
    CONSONANTES ="bcdfghjlmnpqrstvw"
    VOCALES = "aeiou"
    C_DIFICIL = "kxyz"
    nueva = ""
    desordenada = ""

    nueva = dame_4(nueva,CONSONANTES)
    nueva += dame_2(nueva ,VOCALES)
    nueva += dame_1_dificil(nueva,C_DIFICIL)
    desordenada = desordenar_cadena(nueva, desordenada)

    return desordenada

#elige una letra de las letras en pantalla
def dame_letra(letras_en_pantalla):
    for letra in random.choice(letras_en_pantalla):
        return letra
```

def procesar(): Esta función recibe como parámetro la letra_principal , la letra_en_pantalla_candidadata, diccionario y palabras_acertadas.
Primero nos aseguramos mediante un if y la función no_repite que la palabra candidata no se encuentre dentro de la lista de palabras_acertadas, luego verificamos si es_valida() y si esto se cumple retornamos puntos positivos, punto negativo si alguna no se cumple y 0 si se repite.

```python
#si es valida la palabra devuelve puntos sino resta.
def procesar(letra_principal, letras_en_pantalla, candidata, diccionario , palabras_acertadas):
    if no_repite(candidata , palabras_acertadas):
        if es_valida(letra_principal , letras_en_pantalla, candidata ,diccionario , palabras_acertadas):
            return Puntos(candidata)
        else:
            return -1
    return 0

```

def es_valida: Esta función nos da la oportunidad de que la candidata no se repita, que contenga más de 2 letras, que contenga letras que aparecen en pantalla y la letra principal y por último que esta se encuentre en el diccionario.

```python
"""chequea que se use la letra principal, solo use letras de la pantalla y
exista en el diccionario"""
def es_valida(letra_principal, letras_en_pantalla, candidata, diccionario , palabras_acertadas):

    # Verifica si la letra principal está en las letras disponibles en la pantalla
    if letra_principal not in letras_en_pantalla:
        return False

    # Verifica si todas las letras de la palabra candidata están presentes en las letras disponibles en la pantalla
    for letra in candidata:
        if letra not in letras_en_pantalla:
            return False

    # Verifica si la palabra candidata contiene la letra principal
    if letra_principal not in candidata:
        return False

    # Verifica si la palabra candidata está en el diccionario
    if candidata not in diccionario:
        return False

    # Verifica si la palabra candidata ya ha sido ingresada previamente
    if candidata in palabras_acertadas:
        return False

    # Verifica si la cantidad de letras de la palabra es menor a 3
    if len(candidata) < 3:
        return False

    # Si pasa todas las verificaciones anteriores, la palabra es válida
    return True

```

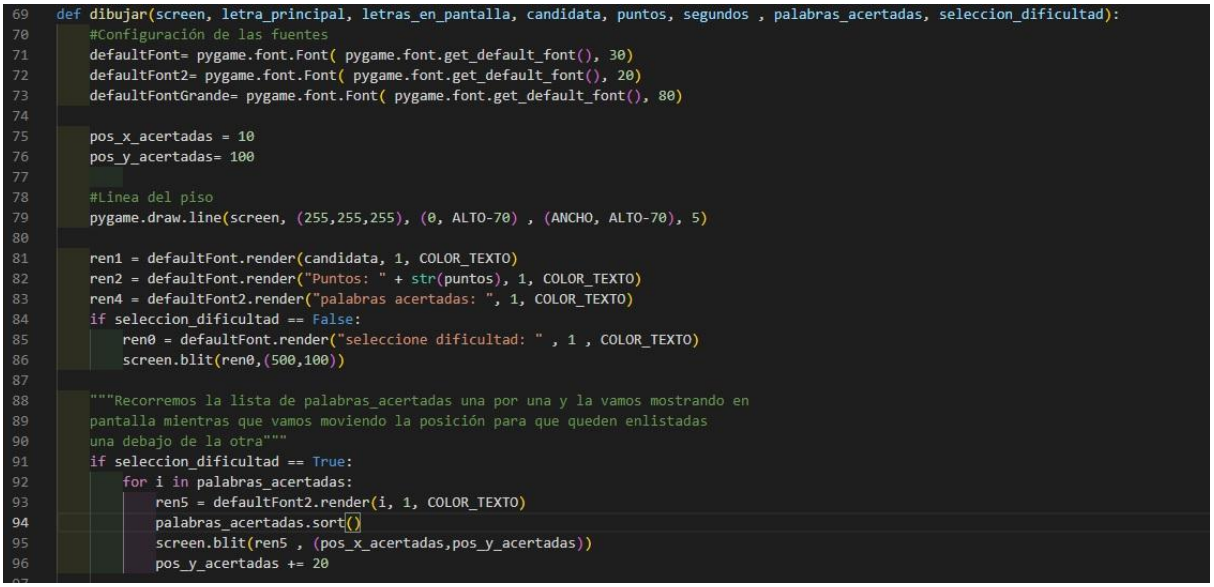
def puntos: Recibe la palabra candidata, y nos retorna la cantidad de puntos dependiendo de la cantidad de letras que contenga la palabra.
def dame_algunas_correctas: Creamos una lista vacía, recorremos cada palabra del diccionario y si la palabras que le pasemos es válida, la guarda en la lista y luego retorna la lista con todas las palabras que son correctas.

```python
#devuelve los puntos
def Puntos(candidata):
    if len(candidata) < 3:
        return 0
    elif len(candidata) == 3:
        return 1
    elif len(candidata) == 4:
        return 2
    elif len(candidata) >= 5 and len(candidata) < 7:
        return (len(candidata))
    elif len(candidata) >=8:
        return (len(candidata))
    elif len(candidata) == 7:
        return 10

#busca en el diccionario palabras correctas y devuelve una lista de estas
def dame_algunas_correctas(letra_principal, letras_en_pantalla, diccionario, palabras_acertadas):
    lista_algunas_correctas = []
    for palabras in diccionario:
        if es_valida(letra_principal, letras_en_pantalla, palabras, diccionario , palabras_acertadas):
            lista_algunas_correctas.append(palabras)
    return lista_algunas_correctas
```

extras.py

De la línea 91 a la 96 nos encargamos de recorrer las palabras acertadas, guardarlas en una variable para luego definir un estilo , ordenarlas alfabéticamente mediante la función sort() , mostrarla en pantalla y mover la posición de la siguiente palabra que se vaya a mostrar.



```
69  def dibujar(screen, letra_principal, letras_en_pantalla, candidata, puntos, segundos , palabras_acertadas, seleccion_dificultad):
70      #Configuración de las fuentes
71      defaultFont= pygame.font.Font( pygame.font.get_default_font(), 30)
72      defaultFont2= pygame.font.Font( pygame.font.get_default_font(), 20)
73      defaultFontGrande= pygame.font.Font( pygame.font.get_default_font(), 80)
74
75      pos_x_acertadas = 10
76      pos_y_acertadas= 100
77
78      #Linea del piso
79      pygame.draw.line(screen, (255,255,255), (0, ALTO-70) , (ANCHO, ALTO-70), 5)
80
81      ren1 = defaultFont.render(candidata, 1, COLOR_TEXTO)
82      ren2 = defaultFont.render("Puntos: " + str(puntos), 1, COLOR_TEXTO)
83      ren4 = defaultFont2.render("palabras acertadas: ", 1, COLOR_TEXTO)
84      if seleccion_dificultad == False:
85          ren0 = defaultFont.render("seleccione dificultad: " , 1 , COLOR_TEXTO)
86          screen.blit(ren0,(500,100))
87
88      """Recorremos la lista de palabras_acertadas una por una y la vamos mostrando en
89      pantalla mientras que vamos moviendo la posición para que queden enlistadas
90      una debajo de la otra"""
91      if seleccion_dificultad == True:
92          for i in palabras_acertadas:
93              ren5 = defaultFont2.render(i, 1, COLOR_TEXTO)
94              palabras_acertadas.sort()
95              screen.blit(ren5 , (pos_x_acertadas,pos_y_acertadas))
96              pos_y_acertadas += 20
```