Informe Bubbly Words

INTRODUCCIÓN:

Este informe fue realizado con el cometido de explicar el desarrollo del Trabajo Práctico que se nos encargó para la asignatura de Introducción a la Programación. Este consistía en un juego con su programa principal ya realizado, y con sus consignas a completar con funciones que ya estaban definidas, y que nosotros debíamos completar para que el programa principal funcione de manera correcta. Este juego tomó el nombre de "Bubbly Words" y a continuación estará toda la información necesaria para entender el funcionamiento de este.

EL JUEGO Y SUS REGLAS:

"Bubbly Words" es un juego que tiene como objetivo encontrar la máxima cantidad de palabras posibles, que se mostrarán en una pantalla, con una suma de letras que estarán en constante descendencia. Esta pantalla estará dividida en tres columnas diferentes, que las conoceremos como primera, segunda y tercera, y por cada columna se tendrá distintos tipos de letra. Esto es importante ya que, como regla del juego, para armar la palabra deseada se debe tener en cuenta el orden de las columnas por su número. Por ejemplo, para armar la palabra, puedo elegir letras de la segunda columna, siempre y cuando no se haya elegido alguna de la tercera. Por otro lado, en este caso, ya no se podrían tomar letras de la primera columna, ya que quedaría "bloqueada" al usar la segunda.

PUNTAJES:

Para el puntaje del juego, se tomó la decisión de darle prioridad a las letras elegidas para construir las palabras. Es decir, que cada letra que conforma la palabra sumara una cantidad de puntos dependiendo que clase de letra sea. En otras palabras, ciertas letras tendrán más puntos que otras, y estarán representadas a continuación:

Vocales: cada vocal en la palabra sumará un total de *1(uno) punto.*

Consonantes: cada consonante en la palabra sumará un total de *2(dos) puntos.*

Consonantes difíciles: las consonantes difíciles serán: "j", "k", "q", "w", "x", "y", "z". Y sumará un total de *5(cinco) puntos por cada una.*

FUNCIONES:

Como fue explicado anteriormente, para el funcionamiento del programa principal, era necesario realizar ciertas funciones previamente definidas en su totalidad. Por lo que a continuación estará la presentación de cada función en el programa, con una breve explicación de su funcionamiento y que rol cumplen en el juego.

cargarLista():

Esta primera función es la que tiene como objetivo elegir la palabra importada desde el lemario de forma aleatoria, y cargarla en cada lista de una columna. En otras palabras, la palabra tomada es “dividida” en tres partes, y luego se reparten de forma de que las primeras letras aparezcan en la primera columna, las del medio en la segunda, y por último mostraría las letras finales en la columna número tres. Por otro lado, también se ocupa de que estas letras aparezcan en la parte alta de la pantalla, para que luego al bajar, tengan su recorrido correspondiente. Esto es gracias a las posiciones que se les asigna de forma medida con cierta altura.

```
# Elige una palabra aleatoria de la lista, la divide y les crea posiciones aleatorias en cada una de las columnas
def cargarListas(lista, listaIzq, listaMedio, listaDer, posicionesIzq, posicionesMedio, posicionesDer):
    # elige una palabra de la lista y la carga en las 3 listas
    # y les inventa una posicion para que aparezca en la columna correspondiente

    # inicializo con una palabra random de la lista
    palabra = lista[random.randrange(0, len(lista))]
    lenght = len(palabra)
    primeraInstancia = lenght//3
    segundaInstancia = 2*(lenght//3)
    fraccion = ANCHO//3
    for i in range(0, lenght):
        alto = random.randrange(30, 40)
        anchoIzq = random.randrange(0+30, (fraccion-30))
        anchoMedio = random.randrange(fraccion+30, ((2*fraccion)-30))
        anchoDer = random.randrange((2*fraccion)+30, (ANCHO-30))
    # Mete las primeras 2 letras en la listaIzq, las 2 siguientes en listaMedio y las restantes en listaDer
        if i >= 0 and i <= primeraInstancia-1:
            if not estaCerca([anchoIzq,alto],posicionesIzq):
                listaIzq.append(palabra[i])
                posicionesIzq.append([anchoIzq, alto])
            else:
                alto = random.randrange(30, 40)
                anchoIzq = random.randrange(0+30, (fraccion-30))
                listaIzq.append(palabra[i])
                posicionesIzq.append([anchoIzq, alto])

        if i >= primeraInstancia and i <= segundaInstancia-1:
            if not estaCerca([anchoMedio,alto],posicionesMedio):
                listaMedio.append(palabra[i])
                posicionesMedio.append([anchoMedio, alto])
            else:
                alto = random.randrange(30, 40)
                anchoMedio = random.randrange(fraccion+30, ((2*fraccion)-30))
                listaMedio.append(palabra[i])
                posicionesMedio.append([anchoMedio, alto])

        if i >= segundaInstancia and i <= lenght-1:
            if not estaCerca([anchoDer,alto],posicionesDer):
                listaDer.append(palabra[i])
                posicionesDer.append([anchoDer, alto])
            else:
                alto = random.randrange(30, 40)
                anchoDer = random.randrange((2*fraccion)+30, (ANCHO-30))
                listaDer.append(palabra[i])
                posicionesDer.append([anchoDer, alto])
```

bajar():

Esta función es llamada para cumplir con la descendencia de las letras mostradas en pantalla. Esto es logrado gracias a que la función llama a las letras a su posición inicial y luego por cada segundo va cambiando de forma negativa en su eje "y" cambiando su altura, haciendo bajar sus posiciones.

```
# Con cada actualizacion baja la posicion de las letras en el eje y
def bajar(lista, posiciones, segundos, dificultad):
    # Hace bajar las letras y las borra cuando llegan a determinada altura
    # El derecho bajaba mas rapido
    for i in range(0,len(lista)):
            posiciones[i][1] = posiciones[i][1] + velocidad(segundos, dificultad)

    for pos in posiciones:
        if pos[1] > 500:
            lista.pop(posiciones.index(pos))
            posiciones.pop(posiciones.index(pos))

    return lista,posiciones
```

Problemas: en el caso de *bajar()* tuvimos la complicación de que al momento de ejecutar el programa, la tercera columna bajaba con más velocidad que las demás.

actualizar():

En este caso, esta es la función encargada de llamar a la función *bajar()* de manera que se muestren las letras en constante movimiento hacia abajo. Por otro lado, esta misma función lleva a cabo la eliminación de letras no usadas al llegar al borde inferior de la pantalla. Esto es porque las letras al llegar a cierta altura verificada por su posición, la función toma que las letras ya no se puedan usar y las quita de la pantalla. Por último, llama a la función *cargarLista()* para actualizar las letras en pantalla.

```
# Hace bajar las letras y carga nuevas aleatoriamente
def actualizar(lista, listaIzq, listaMedio, listaDer, posicionesIzq, posicionesMedio, posicionesDer,segundos,dificultad):
    # Actualiza la posicion de las letras y cada un tiempo aleatorio carga nuevas letras
    # Trantamos de usar los segundos
    dificuladBase = 15
    dificultadExtra = 0
    rand1 = random.randrange(0,dificuladBase - dificultadExtra)
    rand2 = random.randrange(0,dificuladBase - dificultadExtra)
    if rand1 == rand2:
        cargarListas(lista, listaIzq, listaMedio, listaDer, posicionesIzq, posicionesMedio, posicionesDer)
    bajar(listaIzq,posicionesIzq,segundos,dificultad)
    bajar(listaMedio,posicionesMedio,segundos,dificultad)
    bajar(listaDer,posicionesDer,segundos,dificultad)
```

estaCerca():

El cometido de esta función es verificar que las nuevas letras que aparezcan en la pantalla, no figuren encima de otras letras ya mostradas. Esto es para que todas las letras tengan su espacio y se entienda las letras de manera clara. Dicho de otra forma, la función comprueba la posición de las letras ya mostradas y si están fuera del rango de la posición inicial, aparecerán nuevas.

```
# Chequea si hay una letra en la zona de spawn
def estaCerca(elem, lista):
    # Chequea si hay letras dentro del rango de spawn
    # No sabiamos donde iba
    # Crasheaba si usabamos while
    # no aparecian letras de las palabras
    for posLista in lista:
        if elem[0] in range(int(posLista[0])-30, int(posLista[0])+30) and elem[1] in range(int(posLista[1])-30, int(posLista[1])+30):
            return True
    return False
```

Problemas: con *estaCerca() t*uvimos algunos problemas a la hora de ejecutar el programa, ya que este se cerraba al usar "*while*". De igual manera también tuvimos dificultades para que aparecieran las letras de las palabras nuevas cuando actualizaba.

Procesar():

Esta función es la encargada de comprobar si la palabra candidata ingresada es válida, inválida, o repetida en el caso de que ya haya sido usada en juego. Para concluir, más adelante además se le agregó la funcionalidad de que en el caso de que sea una palabra candidata repetida, se resten cierta cantidad de puntos al puntaje del usuario.

```
# Procesar devuelve puntos chequea si se repiten, erran o aciertan palabras, devolviendo los puntos correspondientes
def procesar(lista,candidata, listaIzq,listaMedio,listaDer,posicionesIzq, posicionesMedio, posicionesDer,rachas,repetidas):
    # Chequea con la funcion esValida si la candidata es correcta, incorrecta o repetida y devuelve los puntos correspondientes
    # Podiamos repetir y daba puntos
    if esValida(lista,candidata, listaIzq, listaMedio, listaDer,posicionesIzq, posicionesMedio, posicionesDer,rachas) and candidata not in repetidas:
        repetidas.append(candidata)
        eliminarLetras(candidata,listaIzq,listaMedio,listaDer,posicionesIzq,posicionesMedio,posicionesDer)
        rachaTrue(lista,candidata, listaIzq, listaMedio, listaDer,posicionesIzq, posicionesMedio, posicionesDer,rachas)
        sonidoRachas(lista,candidata, listaIzq, listaMedio, listaDer,posicionesIzq, posicionesMedio, posicionesDer,rachas)
        print(repetidas)
        return Puntos(candidata)
    elif candidata in repetidas:
        return -Puntos(candidata)
    else:
        rachaFalse(lista,candidata, listaIzq, listaMedio, listaDer,posicionesIzq, posicionesMedio, posicionesDer,rachas)
        error = pygame.mixer.Sound("./soundFx/error.mp3")
        error.play()
        return 0
```

Problemas: en esta función tuvimos como dificultad la suma de palabras repetidas. Esto hacía que aunque el usuario ya haya usado una palabra válida, si la volvía a utilizar , se le otorgaban los puntos nuevamente.

esValida():

En este caso, esta función es la encargada de verificar si la palabra candidata ingresada es válida para el juego, de forma que cumpla las reglas antes aclaradas. En otros términos, comprueba que las letras de la palabra hayan cumplido con el orden de las columnas puestas en pantalla. Para que en caso de que no esté dentro de las normas del juego, no la sume al puntaje.

```
# Valida la candidata
def esValida(lista,candidata, listaIzq, listaMedio, listaDer,posicionesIzq, posicionesMedio, posicionesDer,rachas):
    # Al principio hicimos una validacion extraña
    # aveces validaba cuando no habia letras
    candidata = candidata
    lenght = len(candidata)
    candidata1 = ""
    puedoIzq = True
    puedoMedio = True
    puedoDer = True
    for i in range(0,lenght):
        if candidata[i] in listaIzq and puedoIzq:
            indexIzq = listaIzq.index(candidata[i])
            candidata1 += listaIzq[indexIzq]
            if candidata1 == candidata and candidata in lista:
                return True
        elif candidata[i] in listaMedio and puedoMedio and candidata in lista:
            puedoIzq = False
            indexMedio = listaMedio.index(candidata[i])
            candidata1 += listaMedio[indexMedio]
            if candidata1 == candidata:
                return True
        elif candidata[i] in listaDer and puedoDer and candidata in lista:
            puedoMedio = False
            puedoIzq = False
            indexDer = listaDer.index(candidata[i])
            candidata1 += listaDer[indexDer]
            if candidata1 == candidata:
                return True
    if candidata1 not in lista:
        return False
```

Problemas: en función *esValida()* se nos presentó un problema sobre la validación de palabras cuando la pantalla no mostraba las letras correspondientes.

Puntos():

En este caso, esta función está ocupada para declarar cuantos puntos le corresponde a cada letra de la palabra candidata, revisando a qué tipo de letra pertenece. Como ya fue aclarado anteriormente, en este juego las letras tienen diferentes tipos de puntajes. Las vocales tiene el valor de un punto, las consonantes el valor de dos puntos, y por último está la categoría de consonantes difíciles que tienen el valor de cinco puntos. Por lo tanto, esta termina con la suma final de todas las letras de la palabra ya válida.

```python
# Devuelve puntos dependiendo de las caracteristicas de la candidata
def Puntos(candidata):
    # Recibe la candidata y otorga puntos segun sus caracteristicas
    puntaje = 0
    vocales = "aeiou"
    cons_dificil = "jkqwxyz"
    for letra in candidata:
        if letra in vocales:
            puntaje += 1
        if letra not in cons_dificil and letra not in vocales:
            puntaje += 2
        if letra in cons_dificil:
            puntaje += 5

    return puntaje
```

eliminarLetras():

Esta función fue agregada con el cometido de eliminar las letras que usamos de la pantalla. Esto quiere decir, que luego de verificar que la palabra candidata sea correcta, elimine todas las letras que utilizamos en las diferentes columnas para formarla. Esto es para evitar que se vuelvan a utilizar posteriormente.

```
# Elimina las letras, que pertenezcan a la candidata, en la pantalla
def eliminarLetras(candidata, listaIzq,listaMedio,listaDer,posicionesIzq, posicionesMedio, posicionesDer):
    # eliminaba letras demas
    # crasheaba intentaba borrar dos veces la misma palabra
    candidata = candidata
    lenght = len(candidata)
    candidata1 = ""
    puedoIzq = True
    puedoMedio = True
    puedoDer = True
    for i in range(0,lenght):
        if candidata[i] in listaIzq and puedoIzq:
            indexIzq = listaIzq.index(candidata[i])
            candidata1 += listaIzq[indexIzq]
            listaIzq.pop(indexIzq)
            posicionesIzq.pop(indexIzq)
            if candidata1 == candidata:
                return
        elif candidata[i] in listaMedio and puedoMedio:
            puedoIzq = False
            indexMedio = listaMedio.index(candidata[i])
            candidata1 += listaMedio[indexMedio]
            listaMedio.pop(indexMedio)
            posicionesMedio.pop(indexMedio)
            if candidata1 == candidata:
                return
        elif candidata[i] in listaDer and puedoDer:
            puedoMedio = False
            puedoIzq = False
            indexDer = listaDer.index(candidata[i])
            candidata1 += listaDer[indexDer]
            listaDer.pop(indexDer)
            posicionesDer.pop(indexDer)
            if candidata1 == candidata:
                return
```

Problemas: en este caso, se nos presentaron varios problemas, ya que provocaba que se cierre el juego, luego intentaba eliminar la palabra introducida en dos ocasiones, y por último también borraba letras de más que no correspondían a la palabra valida.

sonidoRachas():

El propósito de esta función es la de crear una tanda de rachas con sus sonidos correspondientes. La racha se crea cuando el usuario ingresa una o más palabras validas, de manera continua, en el juego. Esto también ejecuta un sonido al momento en el que se crea la racha, cada racha tiene un sonido diferente. O sea, que cada vez que vaya aumentando la racha del usuario, estos sonidos van a ir cambiando, hasta llegar a una racha de seis palabras correctas de forma seguidas.

```
# Segun la longitud de la racha ejecuta diferentes sonidos
def sonidoRachas(lista,candidata, listaIzq, listaMedio, listaDer,posicionesIzq, posicionesMedio, posicionesDer,rachas):
    lenght = len(rachas)
    if lenght == 1:
        racha1 = pygame.mixer.Sound("./soundFx/racha1.mp3")
        racha1.play()
    elif lenght == 2:
        racha2 = pygame.mixer.Sound("./soundFx/racha2.mp3")
        racha2.play()
    elif lenght == 3:
        racha3 = pygame.mixer.Sound("./soundFx/racha3.mp3")
        racha3.play()
    elif lenght == 4:
        racha4 = pygame.mixer.Sound("./soundFx/racha4.mp3")
        racha4.play()
    elif lenght == 5:
        racha5 = pygame.mixer.Sound("./soundFx/racha5.mp3")
        racha5.play()
    elif lenght >= 6:
        racha6 = pygame.mixer.Sound("./soundFx/racha6.mp3")
        racha6.play()
```

velocidad():

Por último, esta función es llamada para cambiar la velocidad del juego, dependiendo de la dificultad. Esto quiere decir, que según la dificultad que se elija antes de comenzar el juego, la velocidad con la que caen las letras en pantalla, va a ser menor o mayor.

```
# Segun el tiempo y la dificultad seleccionada le da un valor a velocidad diferentes
def velocidad(segundos,dificultad):
    velocidad = 0
    if len(dificultad) == 1:
        if round(segundos) <= 61 and round(segundos) > 40:
            velocidad = 3
        if round(segundos) < 40 and round(segundos) >= 15:
            velocidad = 6
        if round(segundos) < 15:
            velocidad = 10

    elif len(dificultad) == 2:
        if round(segundos) <= 61 and round(segundos) > 40:
            velocidad = 5
        if round(segundos) < 40 and round(segundos) >= 15:
            velocidad = 10
        if round(segundos) < 15:
            velocidad = 15

    elif len(dificultad) == 3:
        if round(segundos) <= 61 and round(segundos) > 40:
            velocidad = 13
        if round(segundos) < 40 and round(segundos) >= 15:
            velocidad = 18
        if round(segundos) < 15:
            velocidad = 25

    return velocidad
```

Decisiones:

Hacia el tramo final, debimos decidir acerca de la implementación de un historial de jugadores, pero desistimos de la idea y optamos por un único récord, el puntaje más alto y mostrar los datos de la partida.