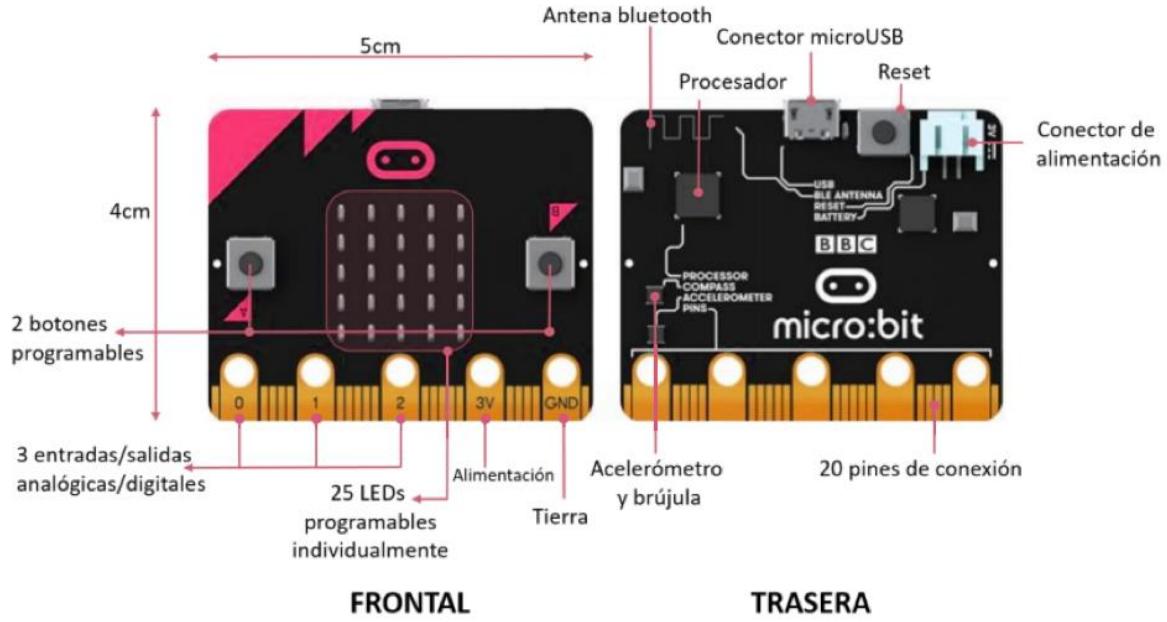
DOCUMENTACION

LED HERO

Índice

Descripción general

LedHero es un juego inspirado en *GuitarHero*, desarrollado para el dispositivo de sistema embebido *Microbit*.



Este realiza animaciones bajando luces en hileras de 5 leds, al llegar la luz abajo se debe presionar un botón, si se presiona a tiempo se lo considera un “hit” o acierto y se ejecutará un tono de una melodía preestablecida. Si no se presiona el botón a tiempo sonará un tono grave que romperá con la armonía de la melodía y omitirá una nota de la misma. Bajarán tantas luces como notas tenga la melodía. Al finalizar la reproducción de esta, se mostrará la cantidad total de aciertos logrados.

Equipo de desarrollo:

Mateo Calvo (Analista)
Lucas González (Analista)
Rubén Ruiz (Diseñador)
Gonzalo Amalfi (Programador y Diseñador)

Tareas de cada área:

Análisis:

Esta área se encargó de analizar la melodía *"Estrellita donde estas",* la cual fue utilizada como base en el juego, buscando encontrar todos los patrones posibles en ella con el fin de facilitar la realización de un algoritmo que pueda facilitar la ejecución de la canción en el juego.
Patrones encontrados:
La melodía repite una sub-melodía 6 veces; esta consta de 7 tonos, primero suenan 2 notas, luego 2 notas distintas y luego otras 2 notas distintas, estas suceden en intervalos de tiempo y duraciones iguales, luego, suena una séptima nota con una duración más larga, la cual, en la primera y quinta ejecución de la sub-melodía esta es la misma que las primeras 2 notas, pero en el resto de ejecuciones es una nota distinta a las otras 6.
La primera y quinta ejecución de la melodía cuenta con 3 notas distintas, y el resto de 4.

Luego también el equipo de análisis diseño bocetos del mapeo que tendría la melodía en el juego. Estas ideas fueron reemplazadas luego por el área de programación por una forma de mostrar la melodía que se generaría aleatoriamente y sería versátil, permitiendo que se pueda reproducir cualquier canción con nuestro programa, se hablará más de esta implementación en la explicación del código.

Diseño:

Esta área se encargó del diseño del logotipo y de la carcasa protectora del *Microbit.*

Programación:

Esta área se encargó del desarrollo del código con el apoyo creativo del equipo de análisis.

Código

Requerimientos funcionales:

Para la realización del programa se piden las siguientes funciones:

Funciones:

- Una función que seleccione aleatoriamente una luz de las 3 del medio, para luego realizar la animación de descenso de la misma.
- Una función que cree la animación de descenso de la luz, hasta la cuarta fila.
- Una función que verifique si se presiona el botón asociado a la luz que está bajando.
- Una función que reemplace la nota actual por un tono grave.
- Una función que modifique el brillo de la luz que desciende.
- Una función que genere el fotograma de la luz en la quinta fila con su nuevo brillo
- Una función que reproduzca la nota correspondiente a la iteración en la que se esté.
- Una función que aumente el contador de hits en 1 si se presiona el botón a tiempo.
- Una función que muestre uno o más fotogramas creados en pantalla.
- Una función que muestre el total de hits logrados en pantalla.

Descripción técnica:

Importación de librerías:

```
from microbit import *
import music
import time
import random
```

microbit contiene las funciones básicas de microbit, **music** es una librería de música de *Microbit* que permite reproducir notas con un parlante conectado a él; se debe poner la letra de la nota, luego la escala de agudo-grave (que va del 1 al 8) que tendrá, y la duración y se debe almacenar como string, por ejemplo, un Do grave, se escribe “C2:2” el primer 2 indica el grave, y el segundo 2 indica la duración que puede tener un valor de entre 1 y 8. **time** contiene la función sleep() y **random** generara un número aleatorio para la selección de la luz.

Definición de las variables configurables:

(Nombre dado internamente, estas son variables cuyos valores pueden ser alterados sin que se corrompa el juego):

```
notesCollection = [
    "C4:2", "C4:2", "G4:2", "G4:2", "A4:2", "A4:2", "G4:7",
    "F:2", "F:2", "E4:2", "E4:2", "D4:2", "D4:2", "C4:7",
    "G4:2", "G4:2", "F4:2", "F4:2", "E4:2", "E4:2", "D4:7",
    "G4:2", "G4:2", "F4:2", "F4:2", "E4:2", "E4:2", "D4:7",
    "C4:2", "C4:2", "G4:2", "G4:2", "A4:2", "A4:2", "G4:7",
    "F:2", "F:2", "E4:2", "E4:2", "D4:2", "D4:2", "C4:7"
]

on = "4"
off = "0"
lateralOfRow = "1"
successLightBrightness = "9"
failureLightBrightness = "2"
failureNote = "C2:2"
presetSleepTimeToReact = 0.3
presetDelayValue = 100
```

Definición de las variables no configurables:

(la modificación de estas si podría corromper el juego:).

```
actualNote, counter, hitsCounter = 0, 0, 0
correctButtonHasPressed = False
startingAnimationFrames, endAnimationFrame, leftLight, middleLight,
rightLight, rowOn, rowOff = str, str, str, str, str, str, str
```

Método selectRandomLight():

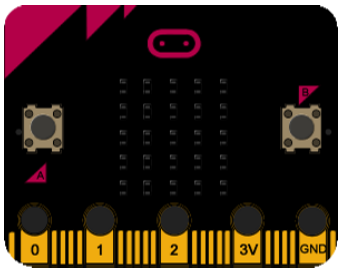
```
def selectRandomLight():
    global leftLight, middleLight, rightLight

    randomLight = random.randint(1, 3)
    if randomLight == 1:
        leftLight, middleLight, rightLight = on, off, off
    elif randomLight == 2:
        leftLight, middleLight, rightLight = off, on, off
    else:
        leftLight, middleLight, rightLight = off, off, on
```

Primero se le da un valor aleatorio en el rango de 1 y 3 a randomLight, y luego se le asigna el valor *on* a una de las 3 luces, y off a las otras 2, según el número que se haya generado.

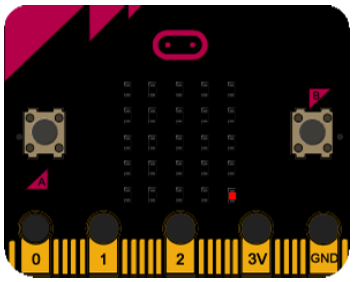
Método createTheDescentAnimationOfSelectedLight():

```
def createTheDescentAnimationOfSelectedLight():
    global startingAnimationFrames, rowOn, rowOff

    rowOn = lateralOfRow + leftLight + middleLight + rightLight + lateralOfRow
+ ":"
    rowOff = lateralOfRow + off * 3 + lateralOfRow + ":"

    frame01 = Image(rowOn + rowOff * 4)
    frame02 = Image(rowOff + rowOn + rowOff * 3)
    frame03 = Image(rowOff * 2 + rowOn + rowOff * 2)
    frame04 = Image(rowOff * 3 + rowOn + rowOff)
    startingAnimationFrames = [frame01, frame02, frame03, frame04]
```

Esta función primero accede a las variables externas *startingAnimationFrames* *rowOff* y *rowOn*, luego le da los valores actuales de las 3 luces del medio de la fila y el valor *lateralOfRow* para cada lateral. Define los 4 frames en la que la luz seleccionada baja de fila en fila. Para entender esto, se debe entender que para generar un frame, por ejemplo, vacío, en *Microbit* se debe definir un objeto de la clase *Image* con un atributo string como este “00000:00000:00000:00000:00000:” ;



entonces, cada 0 corresponde a un led de cada fila, y su valor indica su brillo, 0 es apagado, y luego de 1 a 9 estará encendida con distintas intensidades, y cada “:” es el separador entre fila y fila. O sea que si por ejemplo pasamos como parámetro el string “00000:00000:00000:00000:00009:”



estamos diciendo que esa imagen dejará todos los leds apagados excepto el de la quinta fila, en la última columna, el cual estará encendido a máxima intensidad.
Comprendiendo eso, se entiende la imagen que se está pasando a cada frame. Finalmente, se listan todos los frames en *startingAnimationFrames.*

Método checkIfTheCorrectButtonHasBeenPressed():

```
def checkIfTheCorrectButtonHasBeenPressed():
    global correctButtonHasPressed

    time.sleep(presetSleepTimeToReact)

    if leftLight != off:
        if button_a.is_pressed():
            correctButtonHasPressed = True
        else:
            correctButtonHasPressed = False
    elif middleLight != off:
        if button_a.is_pressed() and button_b.is_pressed():
            correctButtonHasPressed = True
        else:
            correctButtonHasPressed = False
    else:
        if button_b.is_pressed():
            correctButtonHasPressed = True
        else:
            correctButtonHasPressed = False
```

Esta función pausa el programa por *presetSleepTimeToReact* segundos, y luego pregunta si se está presionando el botón respectivo a la luz que no vale *off*; a la luz izquierda le corresponde el botón “a”, a la luz del medio el botón “a” y “b” en simultáneo, y a la luz derecha le corresponde el botón “b”. La respuesta a esa condición if, se almacenará en la variable *correctButtonHasPressed.*

Método changeActualNoteToFailureNote():

```
def changeActualNoteToFailureNote():
    global notesCollection
    notesCollection[actualNote] = failureNote
```

Se reemplaza la nota que se iba a reproducir en la iteración actual, por el tono grave almacenado en *failureNote*.

Método modifyTheSelectedLightBrightness():

```
def modifyTheSelectedLightBrightness():
    global leftLight, middleLight, rightLight

    if correctButtonHasPressed:
        if leftLight != off:
            leftLight = successLightBrightness
        elif middleLight != off:
            middleLight = successLightBrightness
        else:
            rightLight = successLightBrightness
    else:
        if leftLight != off:
            leftLight = failureLightBrightness
        elif middleLight != off:
            middleLight = failureLightBrightness
        else:
            rightLight = failureLightBrightness
```

Primero se consulta el valor de *correctButtonHasPressed*,si es true asigna *successLightBrightness* a la luz

Método createEndAnimationFrame():

```
def createEndAnimationFrame():
    global rowOn, rowOff, endAnimationFrame

    rowOn = lateralOfRow + leftLight + middleLight + rightLight + lateralOfRow
+ ":"
    rowOff = lateralOfRow + off * 3 + lateralOfRow + ":"

    endAnimationFrame = Image(rowOff * 4 + rowOn)
```

Se actualizan rowOn y rowOff con los nuevos valores de las luces, y se crea el ultimo frame que mostrara la luz con más o menos brillo en función de si *correctButtonHasPressed* vale True o False.

Método playActualNote():

```
def playActualNote():
    music.play(notesCollection[actualNote])
```

Se reproduce la nota que se encuentra en la lista de notas *notesCollection*, en la iteración actual.

Método refreshActualNote():

```
def refreshActualNote():
    global actualNote
    actualNote += 1
```

Se aumenta el iterador actualNote en 1.

Método refreshHitsCounter():

```
def refreshHitsCounter():
    global hitsCounter
    if correctButtonHasPressed:
        hitsCounter += 1
```

Se aumenta el contador *hitsCounter* en 1, si solo si, *correctButtonHasPressed* vale *True*

Método displayTotalHits():

```
def displayTotalHits():
    display.scroll("HITS: " + str(hitsCounter))
```

Se muestra el total de *hits* logrados en pantalla.

Game Loop:

```
while counter < len(notesCollection):
    selectRandomLight()
    createTheDescentAnimationOfSelectedLight()

    display.show(startingAnimationFrames, delay=presetDelayValue)

    checkIfTheCorrectButtonHasBeenPressed()
    if not correctButtonHasPressed:
      changeActualNoteToFailureNote()
    modifyTheSelectedLightBrightness()
    createEndAnimationFrame()

    display.show(endAnimationFrame)

    playActualNote()

    refreshHitsCounter()
    counter += 1
```

- Primero se selecciona una luz aleatoria a animar
- Luego se crea la animación
- Se muestra en pantalla
- Se verifica que se presione el botón
- Se reemplaza la nota actual por *failureNote* solo si *correctButtonHasPressed* vale *False*
- Se modifica el brillo de la luz descendiente en relación al valor de *correctButtonHasPressed*
- Se crea el ultimo frame
- Se muestra en pantalla el mismo
- Se reproduce la nota de la iteración actual de la lista de notas
- Se actualiza el contador de *Hits*
- Y por último, se aumenta el contador del Game Loop en 1, para volver a repetir el ciclo, hasta que se cumpla la condición del While.

Por último, se muestran los hits totales en pantalla:

```
displayTotalHits()
```