Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

sl-pr-3

Encapsular el acceso a una aplicación BASIC/MS-DOS

Autor 1:	Gracia Picó, Martina - 795809
Autor 2:	Toral Pallás, Héctor - 798095
Autor 3:	Carrizo Pérez, Daniel - 821674
Grado:	Ingeniería Informática
Curso:	2022-2023

08 de Enero de 2023

Índice

01

ELECCIÓN DEL ENTORNO

Para comenzar y dadas las necesidades de la práctica observamos que al tener una idea similar a la práctica anterior era recomendable usar las mismas herramientas. Entonces usamos Python para la creación del código e hicimos uso de diferentes librerías de Python:

- *Pynput* que nos permite implementar la escritura por teclado como si fuese un usuario.
- *Pygetwindow* que permite la gestión de ventanas.
- *Flask* que permite la creación de un servidor web.
- Otras librerías como *subprocess*, *signal*, *time*, *os* que se usan en momentos concretos de nuestro código para crear subprocesos, gestionar los SIGNAL, poder pausar el código por diferentes periodos de tiempo, y para la gestión de archivos.

El enunciado de la práctica comentaba la posibilidad de realizarla usando *Tesseract*, un OCR que leería la pantalla del emulador y así podríamos saber lo que nos devuelve. Se planteo la idea de realizarla usando OCR, y se comenzó a implementar de está manera, pero debido a lo lento e impreciso que era se buscaron alternativas para la correcta realización de la misma.

02

ENCAPSULACIÓN DE LA APLICACIÓN LEGADA

2.1. INICIO DE LA APLICACIÓN

La aplicación que se utiliza para emular el sistema es MS-DOS es *DOSBox*, la cual nos permite abrir la aplicación *database* que contiene información acerca de casi 800 juegos del ZX Spectrum. Esta se lanza usando un script *.bat* que nos permite pasarle como parámetro la opción *-noconsole* para que no abra una terminal añadida. Una vez lanzado podemos entrar en el navegador "`http://localhost:8080`". Para poder conseguir que la aplicación funcione cliente-servidor funcione había que abrir en el servidor el puerto 8080 para que se pudieran conectar desde un ordenador diferente.

2.2. PANTALLA Y TECLADO

Como ya hemos comentado anteriormente para la gestión del teclado hemos usado la librería *pynput* y con ella hemos implementado una clase Keyboard que contiene las funciones necesarias para la interacción con el emulador.

```
from pynput.keyboard import Controller, Key
class Keyboard:
  def __init__(self): [...]
  def Click_tecla(self, nombre): [...]
  def Escribir_frase(self, frase): [...]
  def Escribir_frase_normal(self, frase): [...]
  def Enter(self): [...]
  def Down(self): [...]
  def Seleccionar_linea(self): [...]
  def Borrar(self): [...]
  def Guardar(self): [...]
  def Seleccionar_todo(self): [...]
```

Para la lectura de la pantalla al principio se uso la biblioteca *pytesseract*, pero como era muy ineficiente, incluso después de haberlo entrenado se buscaron alternativas. Acabamos encontrando una forma de redireccionar la salida del emulador a un fichero que pudiese leer el programa. Tras esto se busco una forma de evitar que este proceso costase tanto tiempo. Leyendo el manual nos dimos cuenta de que había una forma de modificar el número de acciones que el emulador hacía en un ciclo y al modificar esto la velocidad aumento bastante.

Para que estas acciones quedasen automatizadas dentro del programa se intentó implementar un función *modificarCiclosYRedireccion()* que se encarga de modificar la configuración del servidor para que las acciones por ciclo fuese máxima y para que se produjese redirección de la salida.

```
def modificarCiclosYRedireccion():
    configuracion=subprocess.Popen("cd .\\Database-MSDOS\\DOSBox-0.74 && .\\\"DOSBox 0.74
    Options.bat\"", shell=True, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
    while chequearVentana("dosbox-0.74.conf: Bloc de notas")==False: 0
    config = Window("dosbox-0.74.conf: Bloc de notas")
    num_linea = cambiar_ciclos()
    configuracion.wait()
    config.Cerrar_ventana()
    del config
```

Sin embargo al no resultar satisfactoria en todos los ordenadores Windows se tomó la decisión que fuese el usuario quien copiase la nueva configuración en el fichero de texto. La configuración se le daría en un archivo txt y solo tendría que copiar su contenido en el archivo txt que se abre al ejecutar el programa.

2.3. FUNCIONALIDADES IMPLEMENTADAS

Para poder realizar lo solicitado en la práctica se han implementado un código en Python que se encarga de gestionar las diferentes opciones que tiene el emulador.

Para obtener el número de registros que hay en la base de datos simplemente cuando se hace el procesado del fichero obtenemos la cantidad de registros que la base de datos tiene.

```
def procesar(file=archivo):
  # Abre el archivo en modo lectura
  with open(file, "r") as archivo:
    lineas = archivo.readlines() # Lee todas las lineas del archivo
    i = 15
    id = 1
    while (lineas[i].strip()!="1 - INTRODUCIR DATOS"):
      database["datos"].append({"Numero": id,"Nombre": lineas[i].strip(),"Tipo": lineas[i
    +1].strip(),"Cinta": lineas[i+2].strip()})
      database["numReg"] = lineas[i+3].strip()
      i = i + 5
      id = id + 1
```

Para buscar un programa dado su nombre se ha utilizado una función que busca todos los juego que contienen ese nombre puesto que al procesar el fichero creamos un vector que contiene la información de todos los juegos de la database. Esta función devuelve al usuario el listado de programas que contienen dicho nombre.

```
@app.route('/nombre', methods=['POST'])
def nombre_post():
  data = {
    "numReg": database["numReg"],
    "encontrado": "NO",
    "datos": []
  }
  nombre = request.form['nombre'].upper()
  instancias_conversacionales = [instancia for instancia in database["datos"] if nombre in
    instancia["Nombre"]]
  if len(instancias_conversacionales)>0:
    data["encontrado"] = "SI"
    for instancia in instancias_conversacionales:
      data["datos"].append({"numero": instancia["Numero"], "nombre": instancia["Nombre"], "
    tipo": instancia["Tipo"], "cinta": instancia["Cinta"]})
  return render_template("app.html", data=data)
```

Para buscar un programa dada su cinta se ha aplicado la misma metodología que anteriormente pero fijándonos en el campo çinta". Esta función devuelve al usuario un listado de programas que forman parte de dicha cinta.

```
@app.route('/cinta', methods=['POST'])
def cinta_post():
  data = {
    "numReg": database["numReg"],
    "encontrado": "NO",
    "datos": []
  }
  cinta = request.form['cinta'].upper()
  instancias_conversacionales = [instancia for instancia in database["datos"] if cinta in
    instancia["Cinta"]]
  if len(instancias_conversacionales)>0:
    data["encontrado"] = "SI"
    for instancia in instancias_conversacionales:
      data["datos"].append({"numero": instancia["Numero"], "nombre": instancia["Nombre"], "
    tipo": instancia["Tipo"], "cinta": instancia["Cinta"]})
  return render_template("app.html", data=data)
```

Todas esta acciones se han podido realizar porque para redireccionar por salida se ha hecho uso de la opción 6 del emulador que permitía listar todos los juegos de la database.

03

CREACIÓN DEL EJECUTABLE

En esta ocasión al usar las mismas herramientas que en la práctica 2 volvió a surgir el mismo problema a la hora de crear el ejecutable, pero esta vez se implemento el fichero *.bat* directamente sin volver a probar librerías para generar el ejecutable.

3.1. EXPLICACIÓN DE *BUSCATUVIDEOJUEGO.BAT*

Como en la práctica 2 se ha creado este fichero que llama a otros dos fichero .bat con el fin de cumplir el requisito de la práctica en la que se tiene que lanzar la aplicación haciendo doble click. Como esta práctica es cliente-servidor, este archivo simplemente lanza la parte del servidor, puesto que el cliente se tendría que encargar de ingresar en internet a la url que le devuelva el servidor. Se ha tenido en cuenta que la maquina servidor y tiene abierto el puerto que usa la aplicación (8080) para poderse conectar. En caso de que eso no se así consultar el apartado `#^fafd4b`.

BuscaTuVideojuego.bat llama a *launcher.bat* que lanza el servidor de la aplicación y luego llama a *uninstaller.bat* que elimina el entorno virtual.

El archivo *launcher.bat* comienza chequeando si Python está instalado:

```
:: Verificar si Python esta instalado
python --version > nul 2>&1
if %errorlevel% neq 0 (
  echo Python no esta instalado. Por favor, instale Python antes de continuar.
  pause
  exit /b 1
)
```

Luego crea un entorno virtual y lo activa:

```
:: Crear el entorno virtual
echo Creando entorno virtual en %VENV_PATH% si es necesario ...
python -m venv "%VENV_PATH%"

:: Activar el entorno virtual
echo Activando entorno virtual ...
```

```
call "%VENV_PATH%\Scripts\activate.bat"
```

Luego instala las dependencias:

```
:: Instalar las dependencias de tu aplicacion
echo Instalando dependencias si es necesario ...
%PIP% install -r "%REQ_PATH%\requirements.txt" > nul 2>&1
```

Solicita al usuario la modificación de la configuración de DosBox.

```
:: Preparando la configuraci n r pida
echo Perparando la configuracion para un entorno r pido
echo Copia el contenido del fichero COPIAR.txt
timeout /nobreak /t 2 >nul
start notepad.exe ".\BuscaTuVideojuego\COPIAR.txt"
pause
echo Abriendo fichero donde hay que copiar el contenido de COPIAR.txt... (Acuerdate de
    guardar el fichero)
timeout /nobreak /t 2 >nul
cd BuscaTuVideojuego\Database-MSDOS\DOSBox-0.74
DOSBox.exe -editconf notepad.exe -editconf %SystemRoot%\system32\notepad.exe -editconf %
    WINDIR%\notepad.exe
pause
taskkill /IM notepad.exe /F
```

Luego lanza la aplicación pero el .bat sigue ejecutándose:

```
:: Ejecutar la aplicacion Flask y esperar a que termine
cd BuscaTuVideojuego
echo Lanzando Busca tu videojuego ...
start /wait python ".\app.py"
```

El archivo *uninstaller.bat* elimina el entorno virtual creado en *launcher.bat*, si no existiera dicho entorno , no haría nada. Además resetea la configuración de DosBox.

```
:: Eliminar el entorno virtual
echo Eliminando entorno virtual...
rmdir /s /q "%VENV_PATH%"
echo Eliminado

cd BuscaTuVideojuego\Database-MSDOS\DOSBox-0.74
DOSBox.exe -resetconf
```

3.2. CONEXIÓN CLIENTE-SERVIDOR

ˆfafd4b Para que se pueda realizar la conexión cliente-servidor una opción es usar una regla del firewall que nos permita abrir el puerto 8080. Esta opción es la que vamos a explica.

1. Acceder al *Panel de control ¿Sistema y seguridad ¿Firewall de Windows Defender*. Esto se puede hacer o buscando en el buscador de Windows *Panel de control* y siguiendo la ruta.
2. Clicar en *Configuración avanzada*, como es un configuración del administrador es probable que nos pida que solicitemos acceso he incluso la contraseña.
3. Clicar en *Reglas de entrada*.
4. Clicar en *Nueva regla...
5. Seleccionamos *Puerto* como tipo de regla y pulsamos *Siguiente*.
6. Seleccionamos *TCP* y en *Puertos locales específicos* escribimos 8080. Después pulsamos *Siguiente*.
7. Luego escogemos *Permitir la conexión*.

8. En la pantalla siguiente marcamos todas las casillas (*Dominio*, *Privado*, *Público*).
9. Por último, le ponemos un nombre, por ejemplo *A_Práctica_3_SL* y una descripción si queremos.
10. Pulsamos *Finalizar*.

Para habilitarla o deshabilitarla simplemente pulsamos en la regla con el nombre que hemos puesto y nos saldrán las opciones *Habilitar regla* o *Deshabilitar regla*. Es muy importante que cuando no estemos usando dicha regla dejarla deshabilitada para evitar una brecha de seguridad, puesto que esto es una forma rudimentaria de conseguir nuestro objetivo, pero no es el más seguro.

04

IMÁGENES DE MUESTRA

Pantalla principal

Pantalla tras buscar por nombre (Mugsy)

Pantalla tras buscar por cinta (Z)

05

TAREAS Y DEDICACIÓN

Tarea	**Martina**	**Héctor**	**Dani**
Sesión de prácticas	3h	0h	3h
OCR	10h	10h	10h
Implementación de funciones de *keyboard.py* y *window.py*	10min	10min	10min
Implementación de funciones de *app.py*	10h	10h	20h
Creación del fichero "html"	10min	10min	30min
Creación del ejecutable	0min	0min	30min
Total	**23h y 45 min**	**20 h y 20 min**	**34h y 10 min**