Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2021-1

29 de abril de 2021
Actividad Formativa

Actividad Formativa 4

Interfaces Gráficas I

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AF4/`
- **Hora del *push*:** 16:30

Importante: Antes de comenzar, comprueba que Git este funcionando correctamente en tu repositorio privado. Para esto, **sube los archivos base de la actividad de inmediato** (*add*, *commit*, *push*). Se espera que en esta actividad (así como en las demás actividades y tareas) utilices Git a lo largo de **todo tu desarrollo** como una herramienta, no sólo como un método de entrega. Es por esto que recomendamos enfáticamente que vayas subiendo tus cambios constantemente (***push***), ya que **problemas de último minuto** relacionados con la entrega y Git **no serán considerados**.

Introducción

Te levantas, corazón a 100 bpm. y con sudor frío. No has podido dormir bien, recordando cómo el matón del curso te quitó el dinero para el almuerzo... de nuevo. ¡Pero ya no más! Para la próxima llegarás con preparación, pues has decidido a aprender a pelear. Por ello, has creado un programa llamado **DCCarate**, donde podrás entrenar tus habilidades y derrotar al matón la próxima ocasión.

Flujo del Programa

El programa consiste en un juego de pelea, en el que eliges tu personaje y tienes que intentar vencer a tu oponente. Al ejecutar el programa se abrirá la ventana inicial, donde se te pedirá un nombre alfanumérico. Una vez lo ingreses, entrarás a una ventana de selección, donde elegirás al personaje con el que lucharás (se dispone de un **QSpinBox** para ello). Una vez que lo hagas, entrarás a una ventana de combate, donde usando los botones **Patada**, **Frio** y **Defender** deberás intentar derrotar a tu oponente. Finalmente se abrirá la ventana final, donde podrás celebrar tu victoria con buena musica y un lindo GIF (o lamentar tu derrota, pero no pensemos en ese caso).

Archivos

Los archivos relacionados con la interfaz gráfica del programa se encuentran en la carpeta `frontend/`. Esta carpeta contiene a los archivos `ventana_inicio.py`, `ventana_seleccion.py`, `ventana_combate.py` y `ventana_final.py`, en los que se deberán completar los aspectos gráficos del programa donde sea necesario. Además el directorio contiene la carpeta `assets/`, la que tiene los elementos gráficos que se utilizan para complementar las distintas ventanas.

La lógica del programa se encuentra en la carpeta `backend/`. En esta carpeta se encuentra el módulo `logica_menus.py` que se encarga de manejar la lógica de todo el programa. En dicho archivo hay métodos que vienen implementados y otros que deberás completar.

Por último, el archivo `main.py` es el módulo principal del programa, en el que se realiza la conexión de las distintas señales utilizadas para conectar el *front-end* con el *back-end* y se inicializa la aplicación.

Importante: Debido a que en esta actividad se usarán archivos más pesados de lo normal (imágenes y música), te pedimos usar el **.gitignore** que se entrega para ignorar la carpeta `frontend/assets/` y el enunciado. Basta con que copies y pegues el que viene junto al resto de la actividad y **no modifiques su contenido**.

Parte 1: Ventana inicial

La ventana inicial es la primera ventana del programa, ya que permite elegir el nombre de usuario. El usuario debería poder escribir su nombre en un espacio de texto, y presionar un botón que le redirija a la siguiente ventana si es que el nombre cumple los requisitos pedidos (se detallarán estos requisitos más adelante).

Métodos (*front-end*)

El archivo donde deberás trabajar es `frontend/ventana_inicio.py`, en la clase `VentanaInicio`.

- Métodos ya implementados:

- `def __init__(self, ancho: int, alto: int, ruta_logo: str):` Instancia la ventana con sus respectivos atributos. **No modificar**.
- `def recibir_validacion(self, validado: bool):` Este método recibe la aprobación (o reprobación) del *back-end* para ingresar a la siguiente ventana (esto dependerá de si el nombre del jugador cumple con los requisitos o no). En caso de aprobar, se cierra esta ventana y se abre la `VentanaSeleccion`. **No modificar**.

- Métodos que deberás completar:

- `def init_gui(self, ruta_logo: str):` Este método se encarga de crear y ordenar los objetos principales de la ventana. **Deberás crear lo siguiente para completar el método:**
  - Un atributo que contenga el logo del programa. Se recomienda usar las clases `QLabel`, `QPixmap` y el método `setScaledContents` de la clase `QLabel`, junto con `ruta_logo`.
  - Un espacio de texto editable, en donde se pueda escribir el nombre del jugador. Es estrictamente necesario que el nombre del atributo creado sea `self.line_edit_nombre` para que funcione con el resto del código base. Se recomienda usar la clase `QLineEdit`.
  - Una etiqueta que permita saber la utilidad del espacio de texto anterior. En la ventana de ejemplo esta etiqueta corresponde a `"Ingrese su nombre:"`. Se recomienda usar la clase `QLabel`.
  - Un botón que, al ser presionado, permita al programa chequear el nombre y luego pasar a la ventana de selección de personajes. Deberás conectar la señal `clicked` del botón creado al método `enviar_nombre` de esta clase (explicado más adelante).
  - Uno o más *layouts* que permitan ordenar el diseño de la ventana. Los *layouts* deben incluir todos los objetos mencionados anteriormente. Puedes guiarte por el diseño de la ventana de ejemplo aunque no es obligatorio que quede así. Se recomienda usar las clases `QHBoxLayout` y `QVBoxLayout`.
- `def enviar_nombre(self):` Este método deberá emitir la señal `senal_elegir_nombre` al *back-end* para que este verifique que el nombre cumpla con las restricciones. En la señal, la cual ya se encuentra conectada, se debe mandar el nombre que está escrito en el `QLineEdit`[1] creado. **Debes completar este método**.

Métodos (*back-end*)

El backend correspondiente a esta ventana se encuentra en el archivo `backend/logica_menus.py`, en la clase `LogicaInicio`. Esta ya se encuentra implementada, por lo tanto **no la debes modificar.**

- `def __init__(self):` Inicializa el *back-end* de la ventana de inicio. **No modificar**.
- `def comprobar_nombre(self, nombre: str):` Este método recibe el nombre elegido por el usuario (enviado mediante señales desde la ventana de inicio), y determina si lo aprueba o no. Un nombre es aprobado si contiene únicamente caracteres alfanuméricos. Luego, se emite la señal `senal_respuesta_validacion`, que está conectada a `VentanaInicial`, con la respuesta de la aprobación. **No modificar**.

[1] Puedes ocupar el método `.text()` para extraer el texto ingresado de un `QLineEdit`.

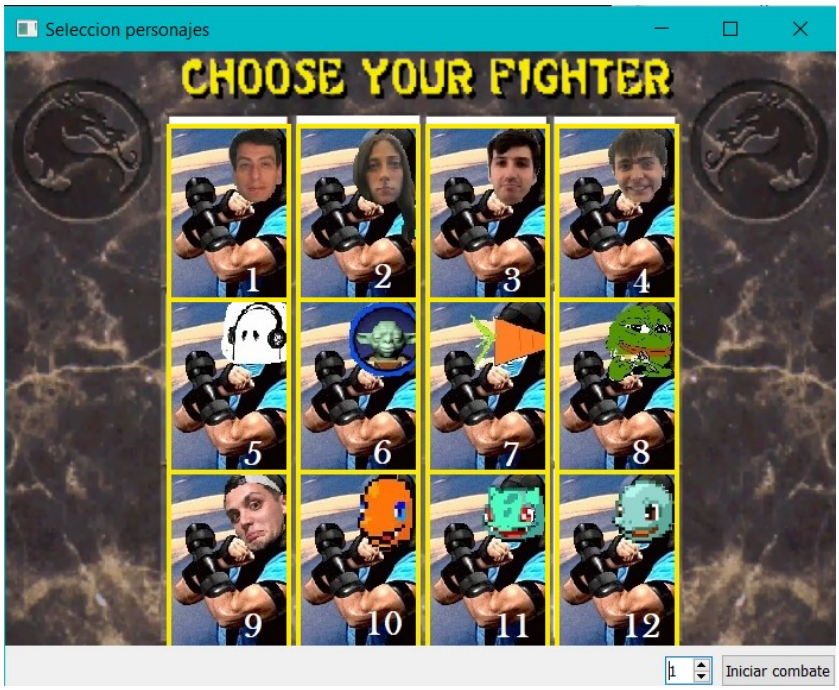
Señales

Las señales que comunican el *front-end* y *back-end* de esta ventana **ya están declaradas y conectadas en el módulo `main.py`**, pero debes tener en cuenta que **debes trabajar con `senal_elegir_nombre`**. Se describirán de todas formas todas las señales solo para que entiendas cómo funciona tu programa:

- `self.senal_elegir_nombre`: Señal emitida por el método `enviar_nombre` de la **VentanaInicial**, en el *front-end*. Envía un `str` con el nombre seleccionado al metodo `comprobar_nombre` del *back-end*, para ver si se aprueba o no. **Esta señal ya está conectada en main.py, sin embargo recuerda que te debes encargar de emitirla en el método `self.enviar_nombre`.**
- `self.senal_abrir_eleccion_personaje`: Señal que se envía un `str` con el nombre seleccionado al método `abrir_ventana` de **VentanaSeleccion**. **Esta señal ya está conectada en main.py**.
- `self.senal_respuesta_validacion`: Señal que envía un `bool` al *front-end*. El `bool` representa la aprobación o reprobación del nombre. **Esta señal ya está conectada en main.py**.

Parte 2: Selección de personajes

En esta ventana deberás seleccionar el personaje con el que desees jugar. Deberás elegir el número correspondiente a tu personaje en un *spinbox*, y luego... ¡comenzar el combate!



Métodos (*front-end*)

El archivo donde deberás trabajar es `frontend/ventana_seleccion.py` en la clase `VentanaSeleccion`.

- Métodos ya implementados:

- `def __init__(self, ancho: int, alto: int, rutas_personajes: list, ruta_fondo: str)`: Este método inicializa la ventana. **No modificar**.
- `def init_gui(self)`: Este método crea los objetos y el diseño de la ventana. **No modificar**.

- `def abrir_ventana(self, nombre: str)`: Este método recibe el nombre entregado por el usuario, lo almacena en el atributo `self.nombre`, y muestra la ventana `VentanaSelección`. **No modificar**.

- Métodos que deberás completar:

- `def iniciar_combate(self)`: Este método se encarga de abrir la ventana de combate y cerrar la de selección, además de enviar la información necesaria para ello. Con este fin, deberás:
  - Extraer el valor encontrado en el `QSpinBox`. Recuerda que este está almacenado en el atributo `self.spinbox_personaje`. Para esto, usa el método `value` de `QSpinBox`.
  - El número extraído del `QSpinBox` está asociado a un personaje en particular, a través del diccionario `self.diccionario_personajes`, con la *key* siendo el valor obtenido previamente y el valor el nombre del personaje.
  - Emitir la señal `self.senal_abrir_ventana_combate` enviando el nombre del usuario (almacenado en el atributo `self.nombre`) y el nombre del personaje (obtenido en el punto anterior), **en ese orden**.
  - Cerrar esta ventana.

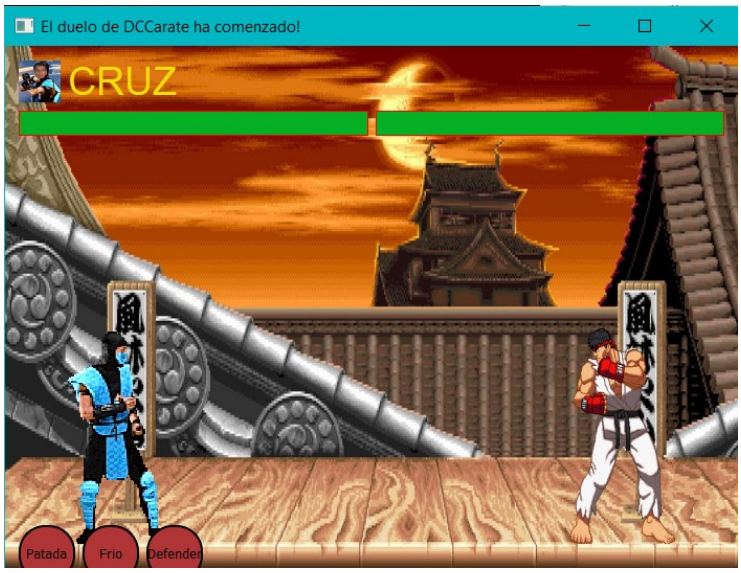
Señales

La señal que comunica el *front-end* y *back-end* de esta ventana **ya está declarada y no debes modificarla, pero aún debes conectarla correctamente** en el módulo `main.py`.

- `self.senal_abrir_ventana_combate`: Esta señal abre la ventana de combate. Además, envía dos `str`, el primero con el nombre del usuario y el segundo con el nombre del personaje seleccionado. **Deberás conectar esta señal en `main.py` con el metodo `combatir` de VentanaCombate, y emitirla desde el *front-end* en el método `iniciar_combate`.**

Parte 3: Ventana de combate

¡Es hora de combatir! Usa los botones **Patada** y **Frío** para hacer daño a tu enemigo, y el botón **Defender** para evitar sus ataques. Pero ojo, que mientras estés defendiendo no podrás atacar!! Deberás dar lo mejor de ti para derrotar a tu temible adversario. En esta parte, deberás conectar los botones a sus métodos respectivos, y bloquearlos o desbloquearlos dependiendo del estado de defensa.



Métodos (*front-end*):

El archivo donde deberás trabajar es `frontend/ventana_combate.py` en la clase `VentanaCombate`.

- Métodos ya implementados:

- `def __init__(self, ancho: int, alto: int, rutas: dict, rutas_iconos: list, estilo_botones: str)`: Inicializa la ventana. **No modificar**.
- `def init_gui(self)`: Estructura la interfaz gráfica de la ventana.**No modificar**.
- `def cambiar_animacion_defensa(self, defendiendo: bool)`: Cambia la animación de la defensa. **No modificar.**
- `def recibir_comando_actualizacion(self, data: dict)`: Recibe un diccionario con la llave çomando". Dependiendo se su *value*, realiza una acción u otra. **No modificar**.
- `def combatir(self, nombre: str, personaje: str)`: Abre la ventana. **No modificar**.
- `def terminar_combate(self, resultado: bool)`: Envía la información a la ventana de resultados de quien gano y esconde la actual. **No modificar**.
- `def frio(self)`: Método que se activa cuando se realiza un **FRIO!**. Envía una señal al *back-end*. **No modificar**.
- `def patear(self)`: Método que se activa cuando se realiza una patada. Muestra en la ventana este ataque y envía una señal al *back-end* para que modifique los valores apropiados. **No modificar**.
- `def enemigo_golpeado(self, vida: int)`: Método que se activa cuando el enemigo es golpeado. Altera la vida de este al nuevo valor en la barra de vida. **No modificar**.
- `def enemigo_prepara(self)`: Cambia el *sprite* del enemigo a su estado de preparación. **No modificar**.
- `def enemigo_deja_preparacion(self)`: Devuelve el *sprite* del enemigo desde su estado de preparación al normal. **No modificar**.
- `def jugador_golpeado(self, vida: int)`: Método que se activa cuando el jugador es golpeado. Altera la vida de este al nuevo valor en la barra de vida y muestra la animación del ataque. **No modificar**.
- `def metodo_defender(self)`: Método que se activa cuando el jugador aprieta el botón defender. Envía una señal al *back-end* informando de esta acción. **No modificar**.
- `def sleep(self, secs: float)`: Inicializa un *timer*. **No modificar**.

- Métodos que deberás completar:

- `def conectar_botones(self)`: Deberás conectar los botones **Patada**, **Frío** y **Defender** con sus métodos respectivos (`patear`, `frio` y `metodo_defender`). **Debes completar este método.**
- `def cambiar_estado_defensa(self, defendiendo: bool)`: Si el jugador esta en acción (determinado por la variable booleana `self.en_accion`), este método no hace nada. En caso contrario, si la variable booleana `defendiendo`, que representa si el jugador está en defensa o no, tiene como valor `True`, deberás bloquear los botones `boton_patada` y `boton_frio`, de esta forma impidiendo que puedas realizar ataques[2]. En caso contrario, deberás volver a activarlos. Además, deberás llamar

[2]Puedes deshabilitar un botón usando el método `setDisabled` de la siguiente manera: `boton.setDisabled(True)`, y puedes habilitarlo usando `False`.

al método `cambiar_animacion_defensa(defendiendo)`. **Debes completar este método.**

Métodos (*back-end*)

El archivo donde deberás trabajar es `backend/logica_menus.py` en la clase `LogicaCombate`.

- Métodos ya implementados:

- `def __init__(self)`: Inicializa el *back-end*. **No modificar**.
- `def recibir_senal(self, comando: str)`: recibe un comando y responde de manera adecuada. **No modificar**.
- `def iniciar_combate(self)`: Inicializa el *timer*[3]. **No modificar**.
- `def detener_combate(self)`: Detiene el *timer*. **No modificar**.
- `def timer_tick(self)`: Aumenta un segundo el *timer*. **No modificar**.
- `def golpear_enemigo(self, tipo: str)`: Método activado cuando el usuario realiza un ataque. Realiza el daño correspondiente y envía una señal al *front-end* notificando los cambios.
- `def defender(self)`: Modifica el estado de defensa del jugador y y envía una señal al *front-end* notificando los cambios.

- Métodos que deberás implementar:

- `def golpear_jugador(self)`: Método activado cuando el enemigo realiza un ataque. Si el jugador no está defendiendo (para esto usa el atributo booleano `self.jugador_defendiendo`) deberás restarle 20 puntos de vida (almacenada en el atributo `self.vida`). Después, deberás emitir la `senal_enviar_actualizacion` con un diccionario, cuyo contenido dependerá de la vida restante del jugador:
  - **Caso 1: La vida restante es menor o igual a cero:**

```
{"comando": "perder"}
```

  - **Caso 2: La vida restante es mayor a cero:**

```
{"comando": "dano_jugador", "valor": self.vida_jugador}
```

  Además, en el primer caso deberás llamar al método `self.detener_combate()` antes de emitir la señal.

Señales

Las señales que comunican el *front-end* y *back-end* de esta ventana **ya están declaradas y conectadas en el módulo `main.py`**, pero debes tener en cuenta que **debes trabajar con `senal_enviar_actualizacion`**. Se describirán de todas formas todas las señales solo para que entiendas cómo funciona tu programa:

- `self.senal_enviar_info_backend`: Envía un (`str`) al *back-end* indicando la acción a realizar. **Esta señal ya está conectada en main.py**.

[3] El concepto de *timer* en PyQt junto con su implementación son contenidos que se verán pronto en el curso.

- `self.senal_abrir_ventana_final`: Esta señal es enviada cuando uno de los dos personajes se queda sin vida. Envía un `bool` indicando si se ganó la partida. **Esta señal ya está conectada en main.py**.
- `self.senal_enviar_actualizacion`: Esta señal envía al *front-end* un diccionario (`dict`) con la llave `comando`, cuyo valor especifica la acción a realizar. Puede o no contener mas llaves dependiendo del comando. **Esta señal ya está conectada en main.py, pero deberás emitirla desde el método `golpear_jugador`**.

Parte 4: Ventana Final

¡La batalla ha terminado! Puedes disfrutar de un lindo GIF y buena música[4] para conmemorar tu victoria (o lamentar tu derrota). El archivo de la ventana final se encuentra en `frontend/ventana_final.py`, contenido en la clase `VentanaFinal`. **No modificar**.

Notas

- Si tienes una pantalla con resolución mayor o igual a 1920x1080 píxeles, puede que las ventanas se vean un poco pequeñas. Esto porque las ventanas se diseñaron para que funcionen correctamente en resoluciones cercanas a los 1280x720 píxeles.
- Recuerda que se entrega un `.gitignore` para que no subas a tu repositorio las imágenes y música. **Este no debe ser modificado**.

Objetivos

- Entender la aplicación de los elementos básicos de interfaces gráficas como Widgets, *Layouts* y señales.
- Implementar ventanas, agregando y ordenando diferentes elementos gráficos de manera apropiada.
- Entender como se comunican las interfaces gráficas en el programa, mediante la creación y uso de señales.

[4]Es posible que el volumen del sonido esté muy fuerte, así que te recomendamos bajar el volumen de tu computador para evitar posibles secuelas.