Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2020-1

15 de Octubre de 2020
Actividad Formativa

Actividad Formativa 03

Interfaces Gráficas I

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AF03/`
- **Hora del *push*:** 16:50

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

¡EMERGENCY MEETING! Debido al éxito de DCCrewmate, los ayudantes para entretenerse han decidido crear **DCCrewmate 2** mediante interfaces gráficas. Sin embargo, "el impostor" ha saboteado el programa, dejando archivos con métodos incompletos. Para recuperar esta actividad deberás crear una ventana, completar algunos métodos y conectar señales, lo que permitirá restablecer el flujo del programa.

Flujo del Juego

Para ayudarte a traer a la vida este juego, se te provee la mayoría del esqueleto del *back-end* y del *front-end* del programa (cada uno en su carpeta respectiva), pero faltan algunas partes que deberás completar. En total son cuatro ventanas, y deberás también conectar señales entre componentes del programa, además de una pequeña parte en el *back-end* de una de ellas. El juego tendrá una ventana de

inicio, una ventana principal y dos ventanas de tareas. Se detallan las funciones de cada ventana y qué partes deberás implementar en las siguientes secciones.

La lógica de juego, es decir, el *back-end* de la aplicación, se encuentra implementado en los módulos de la carpeta `backend/`. En específico, en ese directorio, el módulo `logica_ventanas.py` contiene la lógica de la ventana de inicio y la ventana principal, mientras que en el módulo `tarea_codigo.py` se encarga de manejar la lógica de la tarea 2 - Código. Este último archivo se encuentra incompleto, por lo que deberás completarlo. Estos módulos están implementados de tal forma que se comunicarán con los del directorio *front-end* mediante señales. Todo lo anterior se explican con más detalle a continuación.

La interfaz gráfica del juego se desarrolla en la carpeta `frontend/`. Contiene los módulos `tarea_codigo.py`, `tarea_descarga.py`, `ventana_inicial.py` y `ventana_principal.py`, en los cuales se implementan las cuatro ventanas necesarias para el funcionamiento del juego. También en esta carpeta se te entrega la carpeta `assets/`, la cual contiene las imágenes que se mostrarán en el programa.

El archivo `main.py` es el módulo principal del programa. En este se instancian las clases del *back-end* y los esqueletos de ventanas del *front-end*. Con estas instancias creadas se efectúa la conexión de señales entre ellas, donde algunas ya están hechas, mientras que otras deberás conectarlas tú mismo.

Te recomendamos revisar el código a medida que leas el enunciado, para poder entender bien el flujo del programa.

Consideraciones Generales

El enunciado puede parecer un poco intimidante por su extensión. Sin embargo, no debes preocuparte por ello, ya que la mayor parte de este corresponde a descripción de métodos o señales que ya están implementados. Para cada ventana se tiene una sección de métodos y señales, donde se señala explícitamente al final de ellas qué es lo que debes completar.

Ventana Inicial

La ventana inicial es la ventana que sufrió más daños, por lo que se encuentra prácticamente vacía. Esta es la ventana que da la bienvenida al programa, además de ser donde se ingresa el código para entrar a una partida. Al ingresar un código válido, cualquiera que ya exista en `LISTA_CODIGOS` ubicada en `parametros.py`, se comienza el juego y se pasa a la siguiente ventana. El *front-end* de esta ventana se encuentra en la clase `VentanaInicial`, en el archivo `frontend/ventana_inicial.py`, y una vez completada, debería verse más o menos como en la Figura 1.

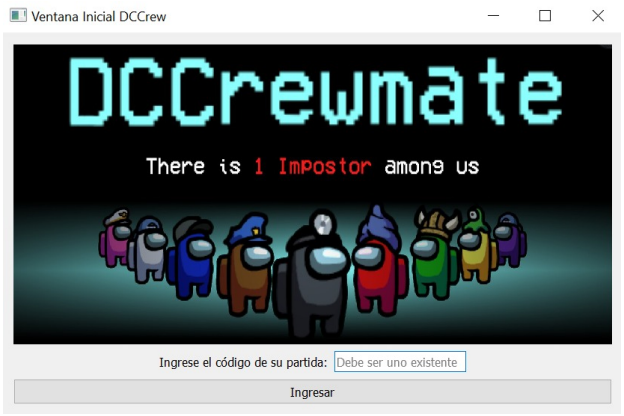

Figura 1: Ejemplo de la Ventana Inicial

Métodos (*front-end*)

A continuación se enumeran y describen todos los métodos del *front-end* de la ventana de inicio. Al final puedes encontrar un resumen de cuáles debes modificar.

- `def __init__(self, ancho, alto, ruta_logo)`: Instancia la ventana. **No debes modificarlo**.
- `def init_gui(self, ruta_logo)`: Crea y agrega los elementos gráficos a la ventana, la cual se encuentra vacía. En ella deberás crear los siguientes elementos:
  - Utilizar un *layout* de forma que el contenido de la ventana quede ordenado de forma similar a la imagen de referencia. No es obligatorio que quede exactamente igual.
  - El logo del programa, cuya ruta es recibida como el argumento `ruta_logo`. Se recomienda darle un tamaño igual al de la ventana. Para esto puedes usar las dimensiones almacenadas en la tupla `self.size` y el método para escalar una imagen, este último lo puede encontrar en los contenidos. Para realizar esta parte te recomendamos utilizar las clases `QLabel` y `QPixmap`.
  - Un campo de texto, donde se pueda ingresar el código para jugar. Este texto ingresado en este *widget* debe ser almacenado en un atributo de instancia llamado `self.input_codigo`. Es importante que tenga ese nombre, ya que de lo contrario no funcionará con el código base. Para realizar esta parte te recomendamos utilizar `QLineEdit`.
  - Un botón para accionar la revisión del código ingresado e intentar ingresar al juego. La señal `clicked` de este botón deberás conectarla al método `self.comparar_codigo`. Puedes utilizar el texto que prefieras en este botón. Para realizar esta parte te recomendamos utilizar `QPushButton`.

En resumen, deberás completar el método `self.init_gui`.

Señales

Las señales que comunican el *front-end* y *back-end* de la ventana de inicio **ya están implementadas y no debes modificarlas**. Además, también **se encuentran conectadas** en el módulo `main.py`. Sin embargo, se describirán de todas formas las más importantes para que entiendas cómo funciona el programa:

- `self.senal_comparar_codigo`: Señal encargada de comunicar el *front-end* con el método que se encarga de revisar el código en *back-end*. Esta emite un *string* (`str`), el que luego se verifica si corresponde o no a uno de los códigos correctos.
- `self.senal_abrir_menu_principal`: Esta señal se emite cuando el código ingresado es correcto y abre la ventana principal de juego.

Ventana de Juego

Una vez validado el código, se abre la ventana de juego, en la que deberás completar algunos métodos. En esta se muestran dos botones, que te llevarán a las dos tareas distintas para realizar. También muestra un contador que se actualiza cuando se completa una tarea.

El *front-end* de esta ventana se encuentra en la clase `VentanaPrincipal`, la que puedes encontrar en el archivo `frontend/ventana_principal.py`. Tu labor será completar los métodos que se señalan a continuación y, una vez terminada esta parte, la ventana debería verse más o menos como en la Figura 2.

Figura 2: Ejemplo de la Ventana de Juego

Métodos (*front-end*)

A continuación se enumeran y describen todos los métodos del *front-end* de la ventana de juego principal. Al final puedes encontrar un resumen de cuáles debes modificar.

- `def __init__(self, ancho, alto, altura_botones, gif_time)`: Instancia la ventana. **No debes modificarlo**.
- `def init_gui(self)`: Se encarga de crear los elementos gráficos base de la ventana. **No debes modificarlo**.
- `def crear_botones(self)`: Este método se encarga de agregar a la ventana los elementos gráficos que interactúan con el usuario para acceder a las tareas. En este deberás crear los botones para ingresar a las tareas y agregarlos al *layout* de la variable `vbox`. Dado que hay dos tareas, deberás crear y agregar dos botones. Sin embargo, la señal `clicked` de ambos deberá estar conectada al método `self.boton_tarea_clickeado`. Por último, el texto de los botones deberá ser **obligatoriamente** `Tarea 1` o `Tarea 2`.
- `def cambiar_diseno_botones(self, boton)`: Al crear los botones en el método anterior puedes llamar a este método, entregándole el botón creado para cambiarle el aspecto al de la imagen del inicio. **No debes modificarlo**.
- `def boton_tarea_clickeado(self)`: Este método emite una señal para abrir la ventana de la tarea 1 o 2 según el botón que recibió el *click*. Al inicio de este método se define la variable `numero_tarea`. Tu labor será utilizar esta variable y si esta tiene el valor de un `int` igual a 1, entonces deberás **emitir** la señal `self.senal_abrir_tarea_codigo`. Por el contrario, si el valor es un `int` igual a 2, entonces deberás **emitir** la señal `self.senal_abrir_tarea_descarga`.
- `def tarea_terminada(self, info_tarea)`[1]: Al inicio de este método se definen dos variables. La primera se llama `numero_tarea`, la cual contiene a un `int` que indica el número de la tarea que fue terminada (1 ó 2). La segunda, se llama `tareas_terminadas` y corresponde a un `int` que indica cuántas tareas han sido terminadas. Tu labor será deshabilitar el botón de la tarea que fue terminada (botón que creaste en el método `self.crear_botones`) con el método `setEnabled(bool)`[2] y actualizar el contador indicando cuántas tareas han sido terminadas del total. Este contador **ya está creado** y se encuentra en el atributo `self.contador_tareas`, correspondiente a una `QLabel`.
- `def terminar_juego(self)`: Este método se encarga de ver si, luego de terminada una tarea, ganas o pierdes. **No debes modificarlo**.

[1] *Hint*: Este método podrás probarlo al completar una de las dos tareas.
[2] Puedes encontrar información de este método en la **documentación correspondiente**.

- `def mostrar_imagen_final(self, es_victoria)`: Este método es el encargado de mostrar el mensaje final luego de obtener tu resultado en `terminar_juego()`. **No debes modificarlo**.

En resumen, deberás completar los métodos `self.crear_botones`, `self.boton_tarea_clickeado` y `self.tarea_terminada`.

Señales

Las señales que comunican el *front-end* y *back-end* de la ventana principal **ya están implementadas y no debes modificarlas**. Además, también **se encuentran conectadas** en el módulo `main.py`. Sin embargo, se describirán de todas formas las más importantes solo para que entiendas cómo funciona tu programa:

- `self.senal_abrir_tarea_descarga`: Señal que se emite al hacer *click* sobre el botón de la tarea 1, abriendo la ventana correspondiente.
- `self.senal_abrir_tarea_codigo`: Señal que se emite al hacer *click* sobre el botón de la tarea 2, abriendo la ventana correspondiente.
- `self.senal_abrir_menu_principal`: Señal conectada al método `self.show` y que se emite cuando una tarea es terminada para volver a mostrar la ventana principal.

Ventana Tarea 1 - Descarga

Una vez terminada la parte anterior, al hacer *click* sobre el botón de la tarea 1 se abrirá su ventana, la que corresponde a la tarea en la que debes hacer *click* sobre un botón para descargar unos datos. El *front-end* de esta ventana se encuentra en la clase `VentanaTareaDescarga` del archivo `frontend/tarea_descarga.py` y, debido a su simplicidad, no tiene *back-end*.

Tu labor será completar algunos métodos del *front-end* que, al terminar, debería verse como en la Figura 3.

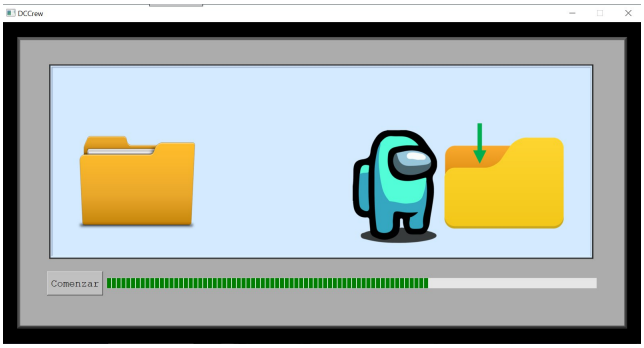

Figura 3: Ejemplo Ventana Tarea 1

Métodos (*front-end*)

A continuación se enumeran y describen todos los métodos del *front-end* de la tarea 1. Al final puedes encontrar un resumen de cuáles debes modificar.

- `def __init__ (self, ancho, alto, tiempo_tarea, tiempo_msg_final)`: Instancia la ventana. **No debes modificarlo**.
- `def init_gui(self)`: En este método debes conectar el botón `self.boton_comenzar_tarea`, mediante su señal `clicked`, al método `self.comenzar_tarea`.

- `def cambiar_diseno_botones(self, botones)`: Se encarga de cambiar el diseño del botón de comenzar tarea. **No debes modificarlo**.
- `def cambiar_diseno_barra_progreso(self, barra_progreso)`: Se encarga de cambiar el diseño de la barra de progreso. **No debes modificarlo**.
- `def comenzar_tarea(self)`: Se encarga de comenzar la tarea, deshabilitar el botón y actualizar la barra de progreso. **No debes modificarlo**.
- `def tarea_terminada(self,)`: Se encarga de mostrar el mensaje final al terminar la tarea y enviar la señal para volver a la ventana de juego principal. **No debes modificarlo**.

En resumen, solo deberás completar el método `self.init_gui`.

Señales

Las señales que comunican el *front-end* y *back-end* de la tarea 1 **ya están implementadas y no debes modificarlas**. Además, también **se encuentran conectadas** en el módulo `main.py`. Sin embargo, se describirán de todas formas las más importantes solo para que entiendas cómo funciona tu programa:

- `senal_abrir_tarea`: Se encarga de mostrar la ventana cuando su botón asociado de la ventana principal de juego es recibe un *click*.
- `senal_tarea_terminada`: Se encarga de, una vez finalizada la tarea, enviar la señal para volver al menú principal y actualizar el contador.

Ventana Tarea 2 - Código

Una vez terminada la parte anterior, al hacer *click* en el botón de la tarea 2 se abrirá su ventana, la que corresponde a la tarea en la que debes ingresar un código aleatorio en el teclado que se muestra. El *front-end* de esta ventana se encuentra en la clase `VentanaTareaCodigo` del archivo `frontend/tarea_codigo.py`, y su *back-end* está en la clase `TareaCodigo` del módulo `backend/tarea_codigo.py`.

Tu labor será completar algunos métodos, tanto del *front-end* como en el *back-end* y, una vez terminada esta parte, la ventana debería verse más o menos como en la Figura 4.

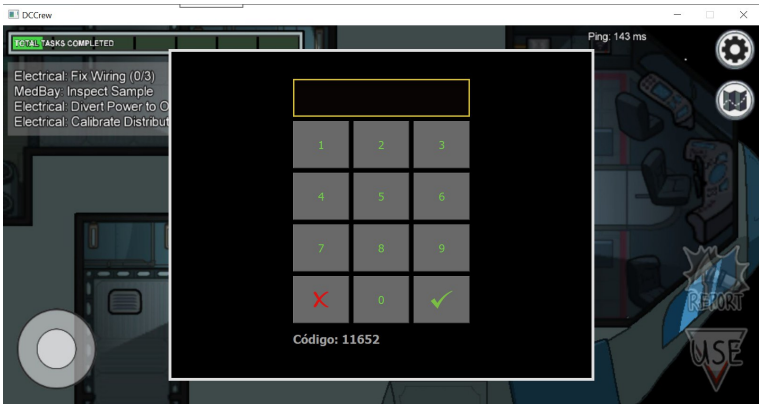

Figura 4: Ejemplo Ventana Tarea 2

Métodos (*front-end*)

A continuación se enumeran y describen todos los métodos del *front-end* de la tarea 1. Al final puedes encontrar un resumen de cuáles debes modificar.

- `def __init__(self, ancho, alto, codigo_correcto, tiempo_msg_final)`: Instancia la ventana. **No debes modificarlo**.
- `def init_gui(self)`: Se encarga de crear los elementos gráficos base de la ventana. **No debes modificarlo**.
- `def cargar_teclado(self)`: Se encarga de crear el teclado de la tarea para insertar el código. **No debes modificarlo**.
- `def cambiar_diseno_botones(self, boton, nombre_boton)`: Se encarga de cambiar el diseño de los botones del teclado y darles un nombre para que las señales reconozcan el tipo de botón. **No debes modificarlo**.
- `def boton_clickeado(self)`: Procesa el mensaje de acuerdo al botón presionado (escribir, borrar, enviar). **No debes modificarlo**.
- `def actualizar_pantalla(self, codigo_actual)`: Este método recibe el argumento `codigo_actual`, correspondiente a un *string*. Tu labor será actualizar el texto de la pantalla del teclado, correspondiente a un `QLabel` almacenado en el atributo `self.pantalla_teclado`.[3]
- `def recibir_comparacion(self, son_iguales)`: Este método recibe un `bool` proveniente del *back-end* que indica si el código ingresado es igual al código correcto. En caso de que sean iguales, es decir, el argumento sea `True`, deberás llamar al método `self.tarea_terminada()`. Por el contrario, si el `bool` es un `False`, entonces deberás actualizar el texto de la pantalla a un *string* vacío. Para esto te recomendamos usar el método implementado en el punto anterior.
- `def tarea_terminada(self)`: Se encarga de mostrar el mensaje final al terminar la tarea. **No debes modificarlo**.

En resumen, deberás completar los métodos `self.actualizar_pantalla`, `self.recibir_comparacion` y `self.tarea_terminada`.

Importante: Para que puedas probar los métodos `boton_clickeado(self)` y `recibir_comparacion(self, son_iguales)` debes completar el método `comparar_codigo(self)` en el *back-end* y conectar la o las señales correspondientes, esto se indica en las siguientes secciones.

Métodos (*back-end*)

A continuación se enumeran y describen todos los métodos del *back-end* de la tarea 2. Al final puedes encontrar un resumen de cuáles debes modificar.

- `def __init__(self, codigo_correcto)`: Instancia el *back-end* de la tarea 2. **No debes modificarlo**.
- `def boton_clickeado(self, tuple_info)`: Procesa la señal de un botón que recibió un *click* de acuerdo al tipo de botón. **No debes modificarlo**.
- `def comparar_codigo(self)`: Este método deberá comparar el atributo `self.codigo_correcto` con `self.codigo_actual`. Si son iguales, deberás asignar el valor `True` al atributo `self.terminado` y emitir la señal `self.senal_resultado_comparacion` con el valor `True`. Por el contrario, si no son iguales, deberás asignar el valor de un *string* vacío al atributo `self.codigo_actual` y emitir la misma señal que antes, pero con un valor `False`.

En resumen, deberás completar el método `self.comparar_codigo`.

[3] Recuerda el método `setText()` de los *widgets*.

Señales

Las señales que comunican el *front-end* y *back-end* de la tarea 2 **ya están implementadas y no debes modificarlas**. Sin embargo hay un problema, **estas no están conectadas**, por lo que será tu labor conectarlas en el módulo `main.py`. Para esto debes tener en cuenta que en este módulo, el *front-end* de la tarea 2 está almacenado en la variable `ventana_tarea_codigo` y el *back-end* en la variable `logica_tarea_codigo`. A continuación se describen las señales que deberás conectar:

- `senal_boton_teclado_clickeado`: Corresponde a una señal perteneciente al *front-end* de la tarea 2 y deberás conectarlo al método `boton_teclado_clickeado` del *back-end* de la misma tarea.
- `senal_nuevo_codigo_actual`: Corresponde a una señal pertenciente al *back-end* de la tarea 2 y deberás conectarlo al método `actualizar_pantalla` del *front-end* de la misma tarea.
- `senal_resultado_comparacion`: Corresponde a una señal pertenciente al *back-end* de la tarea 2 y deberás conectarlo al método `recibir_comparacion` del *front-end* de la misma tarea.

Además, las señales que que se encargan de abrir y cerrar la ventana de la tarea 2, `senal_abrir_tarea` y `senal_tarea_terminada`, **ya están implementadas y no debes modificarlas**. Al igual que en la tarea 1, **se encuentran conectadas** en el módulo `main.py`.

En resumen, deberás conectar en `main.py` las señales `senal_boton_clickeado`, `senal_nuevo_codigo_actual` y `senal_resultado_comparacion`.

Fin del juego

En el módulo `parametros.py` existe un parámetro llamado `PROB_GANAR`. Este es un `float` entre 0 y 1 que define cuán probable es ganar el juego. Una vez terminadas ambas tareas, el programa se encarga de determinar si pierdes o ganas y muestra el mensaje correspondiente. **No debes preocuparte de esto, pues ya está implementado**, aunque puedes jugar con la probabilidad para ver ambos casos.

Objetivos de la Actividad

- Completar ventanas de una interfaz gráfica de acuerdo a especificaciones dadas.
- Conectar correctamente el *front-end* y *back-end* de un programa a través de señales.