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

06 de Mayo de 2021
Actividad Sumativa

Actividad Sumativa 2

Interfaces Gráficas II

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AS2/`
- **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

Como ya sabrás, todos lo años el CAi organiza el Decanazo. Este es un evento en donde estudiantes y profesores podrán encontrarse en un ambiente de distensión y estrechar lazos jugando distintos juegos online. La comisión de Vida Universitaria (quién organiza este magno evento) te pide a ti, admirable estudiante de Programación Avanzada con grandes conocimientos en **Interfaces Gráficas II**, que programes una versión simplificada de **TETRIS** para poder jugar contra **Cristian Ruz** en el próximo evento.

Flujo del Programa

¿Cómo funciona el juego? **TETRIS** deja caer bloques desde la parte superior de la pantalla, y tu objetivo es ir rellenando huecos vacíos (sin bloques) para completar líneas horizontales (puedes ver un pequeño ejemplo en este link). Los bloques que van cayendo los puedes ir moviendo de lado a lado y así vas eligiendo dónde ubicarlos de la manera más conveniente. Una vez que completas una línea horizontal sumas puntos, y esta línea entera se elimina del tablero.

Al ejecutar el programa se abrirá una ventana inicial en donde podrás ingresar un nombre de usuario. Luego, entrarás a la ventana de juego en donde podrás mover los bloques (con las teclas `A` y `D`) y jugar **TETRIS** . Una vez que completes una partida (cuando hayas perdido), se abrirá una ventana con el puntaje que obtuviste.

Archivos

Los archivos relacionados con la interfaz gráfica del programa se encuentran en la carpeta `frontend/`. Esta carpeta contiene los archivos `ventana_inicio.py` y `ventana_juego.py`, los cuales se deberán modificar para completar los aspectos gráficos de **TETRIS** según se pida más adelante. Además, el directorio contiene la carpeta `assets/` la que contiene los elementos gráficos del programa (imágenes, música y archivos `.ui` de Qt Designer).

La lógica del programa se encuentra en la carpeta `backend/`. Esta carpeta contiene a los siguientes archivos `ventana_inicio_backend.py` y `logica_juego.py` los cuales deberás modificar para completar los aspectos de lógica del programa según se pida más adelante.

Además, se entrega el archivo `main.py`, módulo principal que maneja las conexiones de señales entre *fron-tend* y *back-end* utilizadas en el programa, e inicia la aplicación.

Por último se encuentran el archivo `parametros.py`, el cual contiene todos los parámetros fijos del programa así como también las rutas de los distintos componentes gráficos, y el archivo `utils.py`, que contiene una función que genera la representación de las coordenadas de las figuras.

Importante: Debido a que en esta actividad se usarán archivos más pesados de lo normal (imágenes, archivos `.ui` 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 de Inicio

Esta es la ventana de bienvenida a **TETRIS** . Para que funcione debes configurarla de tal manera que permita ingresar un nombre de usuario válido para iniciar una partida y permita salir del juego. Además, deberás crear las señales que se te indiquen y emitirlas. Te sugerimos crear las señales con los nombres indicados en cada método para evitar errores en tu código.[1]

Métodos de *front-end*

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

- Métodos ya implementados:

- `def salir(self)`: Cierra la interfaz. No debes modificarlo

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

- `def mostrar_ventana(self)`: Abre la ventana de inicio cada vez que se inicia una nueva partida. No debes modificarlo

- Métodos que deberás implementar:

Importante: Los atributos `self.boton_comenzar`, `self.boton_salir` y `self.campo_nombre` no los vas a ver en el constructor de la clase, ya que se heredan del archivo generado en Qt Designer. De todas maneras, puedes acceder a ellos con los nombres indicados.

- `def __init__(self)`: Instancia la ventana. Aquí debes conectar el botón para comenzar el juego (`self.boton_comenzar`) con el método `verificar_usuario()`, y el botón para salir del juego (`self.boton_salir`) con el método `salir()`. Debes modificarlo
- `def verificar_usuario(self)`: Este método se encarga de emitir una señal (`senal_verificar_usuario`) al *back-end* con el nombre de usuario ingresado en el campo de texto [2] (`self.campo_nombre`) de la interfaz. Debes crear esta señal. Debes modificarlo

Figura 1: Vista de ventana inicio

Métodos de *back-end*

El archivo en donde deberás trabajar es `backend/ventana_inicio_backend.py`, donde encontrarás la clase `VentanaInicioBackend`. En este archivo, además viene ya implementada la `class Musica` la cual se encarga de reproducir el audio del programa, y que no deberás modificar.[3]

- Métodos ya implementados:

- `def __init__(self)`: Instancia la ventana e inicia acciones del juego como la música. No debes modificarlo
- `def start(self)`: Inicia la música. No debes modificarlo

[2] Puedes acceder al texto del `QLineEdit` del nombre usando el método `.text()` del objeto.

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

- Métodos que deberás implementar:

- `def verificar_usuario(self, usuario: str)`: Recibe el nombre de usuario ingresado desde el *front-end* y verifica si cumple con los requisitos indicados: no debe poseer comas (‘,’) y no debe ser vacío. En caso de que cumpla con los requisitos se debe emitir una señal para empezar el juego (`senal_empezar_juego`) enviando el nombre de usuario. En caso de que el nombre de usuario no cumpla con los requisitos, se deberá emitir una señal para mostrar el mensaje de error (`senal_mensaje_error`). Debes crear las señales. **Debes modificarlo**

Señales

Las señales que conectan el *front-end* y *back-end* de esta ventana se enumeran a continuación. En el ítem anterior debiste crearlas y emitirlas, ahora las debes conectar en el archivo `main.py`

- Señales que deberás conectar:

- `senal_verificar_usuario`: Esa señal es responsable de conectar la `class VentanaInicio` con el método `verificar_usuario` de la `class VentanaInicioBackend`. Envía un `str`. **Debes modificarlo**
- `senal_mensaje_error`: Esa señal es responsable de conectar la `class VentanaInicioBackend` con el método `mostrar` de la `class VentanaError`. No envía parámetros. **Debes modificarlo**
- `senal_empezar_juego`: Esa señal es responsable de conectar la `class VentanaInicioBackend` con los métodos `mostrar_ventana` de la `class VentanaJuego`, `comenzar_partida` de la `class Juego` y `salir` de la `class VentanaInicio` Envía un `str`. **Debes modificarlo**

Parte 2: Ventana de Juego

Esta es la ventana que muestra la interfaz al jugar TETRIS . En esta ventana podrás observar como las figuras empiezan a caer. Deberás mover las piezas con las teclas `A` y `D` para completar filas y así ganar puntos. Si no lo haces las figuras se irán apilando más y más, y cuando lleguen al punto más alto se acabará el juego. `GAME OVER`!!! Debes completar los archivos y métodos según se indique más adelante.

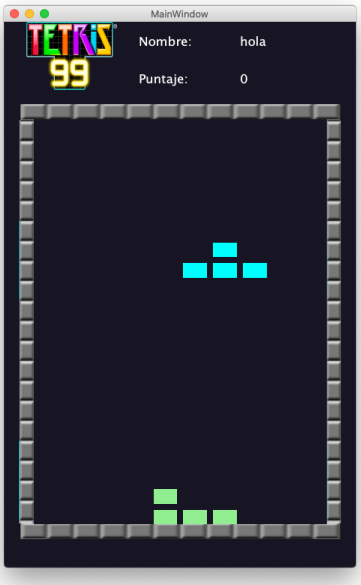

Figura 2: Vista de Ventana Juego

Métodos (*front-end*)

El archivo en donde deberás trabajar es `frontend/ventana_juego.py` en la `class VentanaJuego`.

Métodos ya implementados:

- `def __init__(self)`: Instancia a la ventana. No debes modificarlo
- `def init_gui(self)`: Agrega elementos gráficos a la ventana. No debes modificarlo
- `def armar_diccionario_grilla(self)`: Crea un diccionario con una grilla que contiene a cada casilla del juego. No debes modificarlo
- `def keyPressEvent(self, event)`: Envía señales al *back-end* una vez presionadas las teclas **A** o **D** para mover las figuras. No debes modificarlo
- `def colorear_grilla_entera(self)`: Actualiza los colores de los *labels* de las casillas. No debes modificarlo
- `def fin_juego(self)`: Finaliza el juego una vez que el jugador pierde la partida. No debes modificarlo

Métodos que deberás implementar:

Importante: Los atributos `self.puntaje_usuario` y `self.nombre_usuario`, no los vas a ver en el constructor de la clase, ya que se heredan de Designer. De todas maneras, puedes acceder a ellos con los nombres indicados. Estos atributos son de la clase `QLabel`, lo que deberás considerar a la hora de actualizar sus valores.

- `def mostrar_puntaje(self, puntaje: int)`: Actualiza el puntaje del jugador. Deberás actualizar el puntaje en la interfaz (`self.puntaje_usuario`) con el puntaje recibido como argumento. Debes modificarlo
- `def mostrar_ventana(self, usuario: str)`: Abre la ventana cuando se inicia la partida. Debes inicializar el puntaje del usuario (`self.puntaje_usuario`) en cero y actualizar el nombre de usuario (`self.nombre_usuario`) con el valor recibido como argumento. Estos cambios deben reflejarse en la interfaz gráfica. Debes modificarlo

Métodos (*back-end*)

El archivo en donde deberás trabajar es `backend/logica_juego.py` en la clase `Juego`.

Métodos ya implementados:

- `def __init__(self)`: Instancia el *back-end* del juego. No debes modificarlo
- `def crear_grilla(self)`: Crea la grilla con las casillas del juego. No debes modificarlo
- `def mover_bloque(self)`: Revisa, cada vez que se presiona una tecla, si alguno de los bloques en movimiento ha colisionado con otro bloque existente. No debes modificarlo
- `def enviar_bloque(self)`: Generar nuevos bloques cada vez que se provoque una colisión No debes modificarlo
- `def avanzar_bloques(self)`: Revisa, luego de cada colisión, si el juego continúa o si se ha perdido la partida. No debes modificarlo
- `def revisar_colisiones(self)`: Revisa si existe una colisión entre los bloques en movimiento y algún bloque de la fila de abajo. No debes modificarlo
- `def vaciar_grilla(self)`: Vacia la grilla, modificando el `diccionario_grilla` No debes modificarlo
- `def verificar_linea(self)`: Verificar si hay alguna linea que tenga que ser eliminada. No debes modificarlo

- `def eliminar_linea(self)`: Elimina la línea que ha sido completada y luego llama a los métodos `self.actualizar_grilla` y `self.actualizar_puntaje`. No debes modificarlo

- Métodos que deberás implementar:

- `def comenzar_partida(self)`: Debes llamar al método `self.enviar_bloque()` y crear un `QTimer` que se encargue de avanzar la figura que esté cayendo. Debes conectar el *timer* con el método `avanzar_bloques()`, y debes hacer que tenga una duración de `TIEMPO_AVANCE` (**TIP**: revisa los atributos del constructor) milisegundos. Finalmente, debes iniciar el `QTimer`. Debes modificarlo
- `def actualizar_grilla(self)`: Actualiza el atributo `diccionario_grilla` y lo envía *front-end* . La actualización del diccionario ya está implementada, pero debes emitir la señal (`senal_enviar_grilla`) que contenga al diccionario de la grilla (`self.diccionario_grilla`) para que el *front-end* actualice la grilla. Debes crear la señal. Debes modificarlo
- `def actualizar_puntaje(self)`: Debes aumentar el atributo `self.puntaje` en `PUNTAJE_LINEA` puntos y emitir la señal (`senal_enviar_puntaje`) enviando el puntaje actual (`self.puntaje`). Debes crear la señal. Debes modificarlo
- `def game_over(self)`: Deberás detener el `QTimer` que hace que los bloques avancen. A continuación debes llamar al método `vaciar_grilla`, actualizar a una lista vacía el atributo `self.bloques` y finalmente emitir la señal de término del juego (`senal_game_over`) con el puntaje (`self.puntaje`). Debes crear la señal. Debes modificarlo

Señales

Las señales que conectan el *front-end* y *back-end* de esta ventana se enumeran a continuación. Hay algunas ya implementadas y conectadas. Las otras, que creaste y emitiste, falta conectarlas en el archivo `main.py`

- Señales ya implementadas:

- `senal_teclas`: Esta señal es responsable por conectar la `class VentanaJuego` con el método `mover_bloque` de la `class Juego` envía un `dict`. No debes modificarlo
- `empezar_senal_frontend`: Esta señal es responsable de conectar la `class VentanaJuego` con el método `comenzar_partida` la `class Juego`. No envía nada. No debes modificarlo

- Señales que deberás conectar:

- `senal_enviar_grilla`: Esta señal es responsable por conectar la `class Juego` con el método `colorear_grilla_entera` de la `class VentanaJuego`. Envía un `dict`. Debes modificarlo
- `senal_enviar_puntaje`: Esta señal es responsable por conectar la `class Juego` con el método `mostrar_puntaje` de la `class VentanaJuego`. Envía un `int`. Debes modificarlo
- `senal_game_over`: Esta señal es responsable de conectar la `class Juego` con el método `fin_juego` de la `class VentanaJuego` y el método `mostrar_ventana` de la `class VentanaFin`. Envía un `int`. Debes modificarlo

Parte 3: Ventana Fin de Juego

Esta ventana se abre cada vez que el jugador pierde el **TETRIS** . Debes completar el archivo y método según se indique.

Figura 3: Vista de Ventana Fin de Juego

Métodos (*front-end*)

El archivo en donde deberás trabajar es `frontend/ventana_juego.py` en la `class VentanaFin`.

- Métodos ya implementados:

- `def __init__(self)`: Instancia el *front-end* de la ventana. No debes modificarlo
- `def mostrar_ventana(self, puntaje)`: Actualiza el puntaje del jugador al obtenido en la partida, y muestra la ventana de fin de juego. No debes modificarlo

- Métodos que deberás completar:

- `def emitir_senal_inicio(self)`: Esta señal esconde la ventana de fin de juego, y emite la señal (`senal_inicio`) para redirigir el programa a la interfaz de inicio de juego. Debes crear esta señal. Debes modificarlo

Señales

La señal que conecta el *front-end* de esta ventana y *front-end* de la ventana de inicio debiste crearla y emitirla en el ítem anterior. Ahora debes conectarlas en el archivo `main.py`

- Señales que deberás conectar:

- `senal_inicio`: Esa señal es responsable de conectar la `class VentanaFin` con el método `mostrar_ventana` de la `class VentanaInicio`. No envía nada. Debes modificarlo

Ventana Error

Esta ventana es la que se muestra en caso de que se ingrese un nombre de usuario que no es válido, por lo que se abre cada vez que se emite una señal desde el *back-end* de la ventana de inicio. No debes modificar nada de esta ventana.

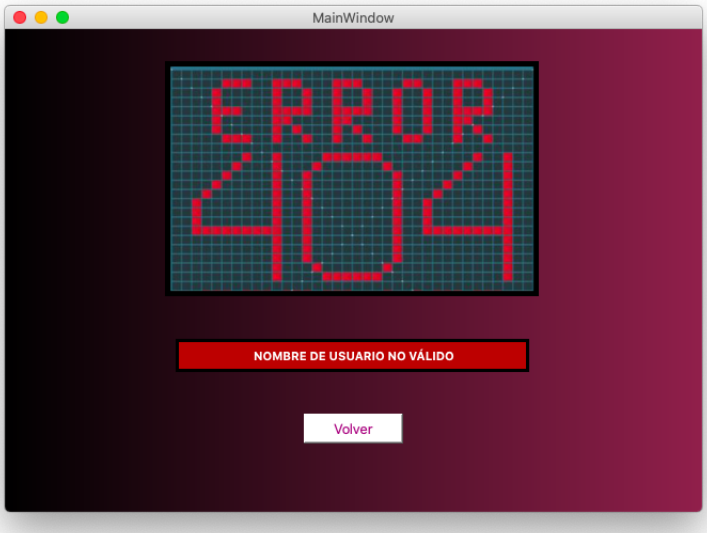

Figura 4: Vista de Ventana Error

Métodos (*front-end*)

El archivo en donde se encuentra esta ventana es `frontend/ventana_inicio.py` en la `class VentanaError`. Los métodos ya vienen implementados y no debes modificarlos.

- Métodos ya implementados:

- `def __init__(self)`: Instancia el *front-end* de la ventana. No debes modificarlo
- `def mostrar(self)`: Abre la ventana de error. No debes modificarlo
- `def esconder(self)`: Esconde la ventana del error. No debes modificarlo

Notas

- La recolección de la actividad se hará en la rama principal (`main`) de tu repositorio.
- La interfaz del programa funciona incluso si no implementas nada, así que puedes testear su funcionamiento de forma interactiva.
- Si aparece un error inesperado, ¡léelo! Intenta interpretarlo.
- Siéntete libre de agregar nuevos `print` en cualquier lugar de tu código para encontrar errores. Es una herramienta muy útil.
- Recuerda especificar sus dudas en el discord, para que podamos ayudar y encontrar las dudas más frecuentes.

Requerimientos

- (2.00 pts) Parte I: Ventana de Inicio
  - (0.60 pts) Completa correctamente la clase `VentanaInicio` (*front-end*).
  - (0.40 pts) Completa correctamente la clase `VentanaInicioBackend` (*back-end*).
  - (1.00 pts) Implementa correctamente las señales `senal_verificar_usuario`, `senal_mensaje_error` y `senal_emepezar_juego`.
- (2.50 pts) Parte II: Ventana de Juego
  - (0.30 pts) Completa correctamente la clase `VetanaJuego` (*front-end*).
  - (1.20 pts) Completa correctamente la clase `Juego` (*back-end*).
  - (1.00 pts) Implementa correctamente las señales `senal_enviar_grilla`, `senal_enviar_puntaje` y `senal_gamer_over`.
- (1.50 pts) Parte III: Ventana Fin de Juego
  - (0.75 pts) Completar correctamente el método `emitir_senal_inicio`
  - (0.75 pts) Implementa correctamente la señal `senal_inicio`