Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2021-2

14 de Octubre de 2021
Actividad Sumativa

Actividad Sumativa 3

Interfaces Gráficas

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AS3/`
- **Hora del *push*:** 16:40

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**.

Importante: Debido a que en esta actividad se usarán archivos más pesados de lo normal (imágenes y archivos `.ui`), deberás completar el archivo `.gitignore` para ignorar la carpeta `frontend/assets/` y el enunciado.

Introducción

Es otra semana, lo que significa otra fecha de partidos en la liga nacional de fútbol. Cruz, como ya es tradición, prende la televisión para ver a su querido Cobreloa... perder otra vez. *Esto no puede seguir*

así... como voy a educar a mis estudiantes a ser ganadores si no tengo un ejemplo a seguir... - murmulló Cruz - *Llego la hora, ¡Tengo que liberar el real potencial que tiene este equipo!.*

Y así, aprovechando la semana de receso, Cruz viajó al Amazonas, navegó vastos océanos, visitó a Santa Claus en el Polo Norte y terminó su viaje en las antiguas pirámides de Egipto. Allí encontró un viejo pergamino con la respuesta a todas sus plegarias.

La DCCobra debe comer
el glorioso símbolo de Cobreloa
y así, y solo así
Cobreloa será DCCobreloa

Flujo del Programa

La aplicación primero abre una ventana de inicio, la cual permite al usuario ingresar su nombre y la contraseña de DCCobra. La ventana de inicio envía la información ingresada por el usuario al *back-end* para poder verificar si los datos son correctos y, en el caso de serlos, empezar el juego. Si la contraseña no es correcta se le pide al usuario reingresarla, sin dejar que empiece el juego.

Cuando el juego comienza se abre la ventana de juego. El juego consta de una (DC)cobra que el jugador puede mover con el teclado. La idea es intentar alcanzar el símbolo de Cobreloa para sumar puntos, los que se mostrarán en pantalla. El juego termina cuando la cobra colisiona contra los límites de la pantalla o contra sí misma. En este caso, se muestra una ventana de fin de juego.

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 DCCobra. La carpeta `assets/` contiene los elementos gráficos del programa (imágenes y archivos `.ui` de Qt Designer). **Debes modificarlo**

La lógica del programa se encuentra en la carpeta `backend/`. Esta carpeta contiene a los archivos `logica_inicio.py` y `logica_juego.py`, los cuales deberás modificar para completar los aspectos de lógica del programa. **Debes modificarlo**

El archivo `main.py`, es el archivo principal, que inicia la aplicación y maneja las conexiones de señales entre *front-end* y *back-end* utilizadas en el programa. **Debes modificarlo**

De manera especial, en esta actividad se incluye un archivo `.gitignore` **que se encuentra vacío**, el cual deberás completar. **Debes modificarlo**

Por último, se encuentra el archivo `parametros.py`, el cual contiene todos los parámetros fijos del programa así como también las rutas de los distintos componentes gráficos. **No debes modificarlo**

Parte 0: Uso de `.gitignore`

Deberás completar el archivo `.gitignore` para **ignorar** la carpeta `frontend/assets/` y el enunciado. Debes agregar **ambas reglas**, aún cuando no tengas el enunciado en tu respositorio local. **Debes modificarlo**

Parte 1: Ventana de Inicio

En esta primera parte, tendrás que implementar la ventana de inicio de DCCobra de tal manera que se permita ingresar un nombre de usuario y una contraseña para iniciar una partida. Para esto, deberás manejar el diseño de la ventana, incluyendo una imagen con el logo del juego, además de conectar y emitir señales entre el *front-end* y el *back-end* de la aplicación.

Figura 1: Vista de ventana inicio

Métodos de *front-end*

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

- Métodos ya implementados:

- `def __init__(self, tamano_ventana: QRect)`: Inicializa la ventana. **No debes modificarlo**
- `def agregar_estilo(self)`: Añade detalles estéticos a la ventana, y permite que se pueda usar `ENTER` para iniciar la sesión. **No debes modificarlo**
- `def recibir_validacion(self, tupla_respuesta: tuple)`: Método que recibe la señal de validación del *back-end*, con la información de si la contraseña ingresada es correcta o no. Según la información ingresada, ocultará la ventana o notificará que la contraseña es incorrecta. **No debes modificarlo**
- `def mostrar(self)`: Muestra la ventana en pantalla. **No debes modificarlo**
- `def ocultar(self)`: Oculta la ventana de la pantalla. **No debes modificarlo**

- Métodos que deberás implementar:

- `def init_gui(self, tamano_ventana: QRect)`: Este método parte asignando el ícono de la ventana, lo que ya esta implementado (no debes modificar esta linea). Luego, deberás crear y ordenar los diferentes elementos de la ventana de inicio. Esta parte podrás realizarla de la manera que más te acomode, sin embargo **debe contar con los siguientes elementos:** **Debes modificarlo**
  - La ventana de inicio debe tener un tamaño definido, para esta parte se debe hacer uso del argumento **`tamano_ventana`**, el cual es de tipo `QRect`.

- Debe contar con un `QlineEdit` para ingresar el nombre de usuario. Este espacio deberá llamarse `self.usuario_form`. Es muy importante que en la interfaz se indique explícitamente para qué es este espacio, por ejemplo, mediante un `QLabel`.
- Debe contar con un `QLineEdit` para ingresar la contraseña. El `QlineEdit` **debe llamarse** `self.clave_form`. Después de crearlo, debes utilizar su método `setEchoMode` que recibe como argumento exacto `QLineEdit.Password`[1]. La interfaz debe indicar explícitamente para que es este espacio, por ejemplo, mediante un `QLabel`.
- Debe mostrar el logo preparado para esta actividad. Para esta parte deberás hacer uso del parámetro `RUTA_LOGO`. Te recomendamos ajustar el tamaño del logo para que se vea bien en la ventana (un buen tamaño es hasta 400x400 Para esto puedes usar el método `setMaximumSize`).
- Debe incluir un botón que permita enviar el nombre de usuario y la contraseña. El botón **debe llamarse** `self.ingresar_button`, y debe estar conectado con el método `self.enviar_login`.
- Finalmente, debes llamar al método `self.agregar_estilo`, para añadir estética y otras funcionalidades extra.

- `def enviar_login(self)`: En este método, tendrás que hacer uso de la señal `senal_enviar_login` para enviar el usuario y la contraseña ingresada en la ventana como una tupla. Los elementos de la tupla debe encontrarse en ese orden mencionado. **Debes modificarlo**

- Señales de la clase `LogicaInicio`:

- `senal_enviar_login`: Señal encontrada en la clase `VentanaInicio`. Está conectada con el método `comprobar_contrasena` de la instancia de `LogicaInicio`. Al emitirse, envía una `tupla`. Se utiliza para comprobar que la contraseña es correcta cuando se intenta ingresar al juego. **No debes modificarlo**

Métodos de *back-end*

El archivo en donde deberás trabajar es `backend/logica_inicio.py`, donde encontrarás la clase `LogicaInicio`.

- Métodos que deberás implementar:

- `def comprobar_contrasena(self, credenciales: tuple)`: Este método recibe una tupla con nombre `credenciales`, la cual contiene dos `str` con la información del usuario enviada por el método `enviar_login` (nombre de usuario y contraseña, respectivamente). En este método, deberás verificar si la contraseña recibida coincide con la contraseña definida en los parámetros como `CONTRASENA`, sin diferenciar mayúsculas[2]. **Debes modificarlo**
  - En caso de que la contraseña **sea válida** (y solo en este caso) deberás emitir la señal `senal_abrir_juego`, enviando como argumento el nombre de usuario que se ingresó.
  - Luego, independientemente de si la contraseña era válida o no, deberás emitir la señal `senal_respuesta_validacion`. Al emitirse, envia como argumento una tupla con el nombre de usuario como primer valor y un `bool` de segundo valor, que indique si el intento fue válido o no (`True` si la contraseña calzaba, `False` en el caso contrario).

- Señales de la clase `LogicaInicio`:

Esta clase cuenta con señales, las cuales deberás conectar en con sus métodos respectivos en el archivo `main.py`.

[1] Este método hace que al introducir texto, aparezca censurado, como un formulario de contraseña.
[2] Esto quiere decir, que si la contraseña fuera `"Aa"` debería coincidir con `"aa"`.

- `senal_respuesta_validacion`: Señal encontrada en la clase `LogicaInicio`. Debes conectarla con el método `recibir_validacion` de la instancia de `VentanaInicio`. Toma como argumento una `tuple`. **Debes modificarlo**
- `senal_abrir_juego` Señal encontrada en la clase `LogicaInicio`. Debes conectarla con el método `mostrar_ventana` de la instancia de `VentanaJuego`. Toma como argumento un `str`. **Debes modificarlo**

Parte 2: Ventana de Juego

En esta ventana deberás ayudar a la DCCobra a buscar a Cobreloa, usando las cuatro flechas (**WASD**). Pero cuidado... ¡Si chocas con los bordes, o contigo misma, vas a perder! Para ello, deberás implementar y hacer uso de las clases y métodos descritos a continuación.

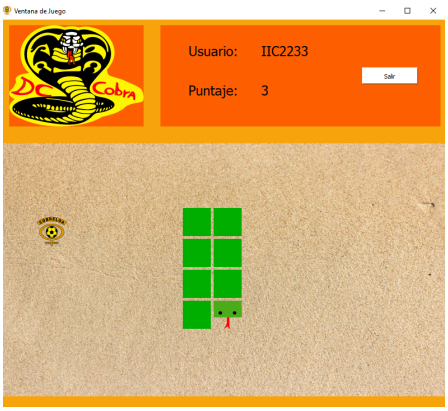

Figura 2: Vista de Ventana Juego

Métodos de *front-end*

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

- Métodos ya implementados:

- `def init_gui(self)`: Agrega elementos gráficos a la ventana. **No debes modificarlo**
- `def iniciar_serpiente(self)`: Agrega los diferentes elementos que conforman a la serpiente. **No debes modificarlo**
- `def cambiar_puntaje(self, puntaje: int)`: Actualiza el puntaje del jugador. **No debes modificarlo**
- `def fin_del_juego(self)`: Ejecuta y muestra la ventana de fin de juego. **No debes modificarlo**
- `def salir(self)`: Cierra la ventana de juego. **No debes modificarlo**
- `def actualizar(self, dic: dict)`: Llama a las funciones encargadas de actualizar los elementos de la ventana. **No debes modificarlo**
- `def avanzar_cobra(self, rect: QRect, direccion: str)`: Actualiza la posición de los diferentes elementos de la serpiente. **No debes modificarlo**

- Métodos que deberás implementar:

En primer lugar, deberás hacer que la clase cargue correctamente el archivo `ventana_juego.ui` (ubicado en la carpeta `frontend/assets`) que fue generado en Qt Designer. La ruta a este archivo se encuentra en el parámetro `RUTA_UI_VENTANA_JUEGO`. **Debes modificarlo**

Luego, deberás **descomentar las 4 conexiones de señales**, que se encuentran bajo el apartado "Ventana Juego", en el archivo `main.py`. Debes modificarlo

- `def __init__(self)`: Instancia a la ventana. Deberás llamar al método que inicializa la interfaz contenida en el archivo `.ui` generado con `Qt Designer`, y luego llamar al método `init_gui` Debes modificarlo
- `def mostrar_ventana(self, usuario: str)`: Método llamado para mostrar la ventana de juego. Deberás primero mostrar la ventana, luego indicar en la interfaz el nombre del usuario y el puntaje con que se parte (0 puntos), y finalmente emitir la señal `senal_iniciar_juego`. El nombre y el puntaje los deberás actualizar en los `QLabel` `self.casilla_nombre` y `self.casilla_puntaje`, respectivamente. Debes modificarlo
- `def keyPressEvent(self, event: QKeyEvent)`: Envía señales al *back-end* una vez presionadas cualquiera de las teclas correspondientes a `TECLA_ARRIBA`, `TECLA_IZQUIERDA`, `TECLA_ABAJO`, `TECLA_DERECHA`, (inicialmente estas corresponden a las teclas `WASD`, pero si lo prefieres las puedes cambiar en `parametros.py`). Para esto deberás emitir la señal `self.senal_tecla` entregándole como parámetro un *string* con la letra que indica la dirección a moverse (`"U"` para moverse hacia arriba, `"L"` para moverse a la izquierda, `"D"` para moverse hacia abajo y `"R"` para la derecha). Debes modificarlo

Métodos de *back-end*

El archivo en donde deberás trabajar es `backend/logica_juego.py` en las clases `LogicaJuego` y `DCCobra` (esta última ya implementada).

`class LogicaJuego:`

- Métodos ya implementados:

- `def __init__(self, cobra: DCCobra)`: Instancia la clase e inicializa sus atributos. No debes modificarlo
- `def detener_juego(self)`: Método encargado de detener el timer al perder. No debes modificarlo
- `def timer_tick(self)`: Método que realiza las distintas acciones que realiza el programa en cada "paso": Avanza al subtick, realiza el avance de la cobra y cheque si hubo alguna colisión, ya sea con los bordes, la cobra misma, o si se atrapó el item. Finalmente, envía una señal al front-end que contiene un diccionario con los atributos actualizados. No debes modificarlo
- `def colision_con_bordes(self)->bool`: Revisa si la cabeza de la cobra no choca con algún borde del mapa. No debes modificarlo
- `def recolecta_item(self)->bool`: Revisa si la cabeza de la cobra se intersecta con el item a atrapar. No debes modificarlo
- `def elegir_posicion_item(self)`: Este método es llamado cuando se atrapa el item (o pasa cierto tiempo sin que se atrape). Elige una posición aleatoria dentro del mapa para ubicarlo, y revisa que no este intersectando con la cobra. No debes modificarlo

- Métodos que deberás implementar:

- `def instanciar_timer(self)`: En este metodo deberás instanciar un `QTimer` y conectarlo al método `self.timer_tick`. Además, deberás asignar que el tiempo entre ejecuciones del método sea de medio segundo. Finalmente, deberás crear el atributo `self.subtick` e inicializarlo en 0. Debes modificarlo
- `def iniciar_juego(self)`: En este método deberás iniciar el `Qtimer` que instanciaste en el método anterior. Debes modificarlo

`class DCCobra:`

Esta clase controla lo relacionado al movimiento de la DCCobra, su pocision y revisar si choca consigo misma. Ya se encuentra completamente implementada.

- Métodos ya implementados:

- `def __init__(self)`: Inicializa la cobra. No debes modificarlo
- `def cambiar_direccion(self, nueva_direccion: str)`: Cambia la dirección en la que se dirige la cobra, teniendo en cuenta que no se puede mover en la dirección contraria a la que se está moviendo (por ejemplo, si se está moviendo a la derecha solo podrá cambiar su dirección hacia arriba o abajo, no hacia la izquierda). No debes modificarlo
- `def avanzar(self)-> bool`: Actualiza la posicion de cada bloque de la cobra, y revisa si en esta nueva posición se produce algún choque consigo misma. Luego retorna un `bool` indicando si se produce este choque. No debes modificarlo
- `def revisar_autocolision(self) -> bool`: Revisa si el bloque correspondiente a la cabeza de la serpiente choca contra algún bloque de su cuerpo, y retorna un `bool` indicando si se produca este choque. No debes modificarlo

Señales:

En esta parte se usan las siguientes señales para conectar el *front-end* con el *back-end*:

- Señales ya implementadas:

- `senal_perder`: Señal emitida por `LogicaJuego` para avisar al *front-end* que el juego ha finalizado. No envía nada. No debes modificarlo
- `senal_actualizar`: Señal emitida por `LogicaJuego` que envía un diccionario con los nuevos valores de posicion de la cobra, si se atrapó item, y puntaje. No debes modificarlo
- `senal_iniciar_juego`: Señal emitida por `VentanaJuego` para avisar al *back-end* a la función `self.inicar_juego` que el juego ha iniciado. No envía nada. No debes modificarlo
- `senal_tecla`: Señal emitida por `VentanaJuego` que envia un `str` con la letra que le corresponde a cada tecla presionada. Se envía al *back-end* a la clase `DCCobra` al método `self.cambiar_dirección`. No debes modificarlo

Parte 3: Ventana de Derrota

Una vez que la DCCobra choca contra un borde o con ella misma, se acabará el juego y se abrirá la ventana de derrota (no hay ventana de victoria porque es físicamente imposible que Cobreloa gane). En esta parte no debes modificar nada.

Figura 3: Vista de ventana de derrota.

Notas

- La recolección de la actividad se hará en la rama principal (`main`) de tu repositorio.
- 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 tus dudas en el Discord, para que podamos ayudar y encontrar las dudas más frecuentes.

Requerimientos:

- (0.5 pts) Uso de `.gitignore`:
  - (0.50 pts) Completa correctamente el archivo `.gitignore`.
- (3.00 pts) Parte I: Ventana de Inicio:
  - (1.25 pts) Completa correctamente el método `init_gui`.
  - (0.75 pts) Completa correctamente el método `enviar_login`
  - (0.50 pts) Completa correctamente el método `comprobar_contrasena`
  - (0.50 pts) Completa correctamente la conexión de señales.
- (2.5 pts) Parte II: Ventana de Juego
  - (0.25 pts) Completa correctamente la carga del archivo `.ui` y descomentar conexiones pedidas.
  - (0.25 pts) Completa correctamente el método `__init__`.
  - (0.50 pts) Completa correctamente el método `keyPressEvent`.
  - (0.50 pts) Completa correctamente el método `mostrar_ventana`.
  - (0.50 pts) Completa correctamente el método `instanciar_timer`.
  - (0.50 pts) Completa correctamente el método `iniciar_juego`.