Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2022-2

29 de Septiembre de 2022
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`), es importante el uso correcto del archivo `.gitignore` para ignorar la carpeta `frontend/assets/` y el enunciado.

Introducción

Como forma de animar al estudiantado de Computación, el DCC decidió comprar una máquina Arcade y dejarla a libre uso en la sala Álvaro Campos. Luego de escuchar esta noticia, decides ir rápidamente a dicha sala a jugar, pero te topas con que todo el mundo pensó igual que tu y ahora hay una fila enorme. Con todas esas ganas de jugar y no querer hacer esa fila enorme, decides poner a prueba tus conocimientos de PyQt5 e implementar uno de los muchos juegos que tiene la Arcade: DCCubitos. Este juego consiste en controlar una plataforma que permite hacer rebotar una pelota. Solo con dicho control, debes hacer que la pelota choque con cada cubito del juego y los destruya dentro de un tiempo acotado.

Flujo del Programa

El programa comienza con una ventana de inicio, introduciendo el juego al usuario. En esta ventana se debe registrar un nombre de usuario y una contraseña. Si ambos componentes cumplen con los requerimientos solicitados, se pasa a la ventana de juego. En esta ventana el jugador debe presionar las teclas habilitadas para mover una plataforma, de izquierda a derecha, evitando que la pelota caiga debajo de la ventana, con el objetivo de que ésta rompa los bloques. Cada vez que estos son destruidos, se sumarán puntos en el juego y por cada pelota que caiga debajo de la ventana, se descontarán puntos. El juego termina al momento de destruir todos los bloques, cuando se agote el tiempo o no queden pelotas. Tras esto, se pasará a la ventana post-juego donde se entrega el puntaje final y se dará la opción de volver a jugar o de salir.

Archivos

Los archivos relacionados con la **interfaz gráfica** del programa se encuentran en la carpeta `frontend/`, estos son:

- `ventana_inicio.py`: contiene la clase `VentanaInicio`.
- `ventana_juego.py`: contiene la clase `VentanaJuego`.
- `ventana_postjuego.py`: contiene la clase `VentanaPostjuego`.

Además, en `frontend/` podrás encontrar los elementos gráficos del programa en la carpeta `assets/` (imágenes y archivos `.ui` de *Qt Designer*)

Los archivos relacionados con el **funcionamiento del juego** se encuentran en la carpeta `backend/`, estos son:

- `elementos_juego.py`: contiene las clases `Pelota`, `Plataforma` y `Bloque`.
- `logica_inicio.py`: contiene la clase `LogicaInicio`.
- `logica_juego.py`: contiene la clase `LogicaJuego`.

Adicionalmente se tienen:

- `main.py`: Es el archivo principal que corre la aplicación. **Este es el archivo que debes correr** para comenzar la ejecución del juego.
- `dccubitos.py`: contiene la clase `DCCubitos` que representa la aplicación y maneja las conexiones de señales entre *frontend* y *backend* utilizadas en el programa.
- `parametros.py`: contiene todas las configuraciones fijas del programa así como también las rutas de los distintos componentes gráficos.

Parte 0: Uso de `.gitignore` Importante

Para esta actividad está incluido el `.gitignore` que deben utilizar para poder ignorar correctamente la carpeta `frontend/assets` y el enunciado. Es muy importante asegurarse de que el `.gitignore` esté en el repositorio **ANTES de realizar el primer *push*** para que no se suban las imágenes y los archivos `.ui`.

Parte 1: Ventana de inicio

En esta primera parte, tendrás que implementar la ventana de inicio de DCCubitos. Te dejamos libertad creativa pero debes incluir los siguientes elementos: el logo (cuya ruta está guardada como `RUTA_LOGO` en el archivo `parametros.py`), un campo de texto para ingresar un nombre de usuario, una etiqueta que diga "Ingresa tu nombre de usuario", un campo para poner la contraseña, otra etiqueta que diga "Ingresa tu contraseña" y un botón para ingresar al juego. Además, debes implementar la lógica que permite comprobar que los datos ingresados sean correctos y conectar y emitir las señales para comunicar el front-end con el back-end.

Figura 1: Vista ejemplo de ventana de inicio

Métodos de front-end:

Deberás trabajar en la clase `VentanaInicio` dentro del archivo `frontend/ventana_inicio.py`

- **Métodos ya implementados**
  - No modificar `def __init__(self) -> None:`
    Inicializa la ventana de inicio y llama al método `crear_elementos(self)`.
- **Métodos que deberás implementar**
  - Modificar `def crear_elementos(self) -> None:`
    Este método agrega todos los elementos visuales e interactivos a la ventana. Dentro de esta debes crear:
    - Un **`QLabel`** que contenga el logo cuya ruta está en `RUTA_LOGO`.
    - Un **`QLineEdit`** para ingresar el nombre de usuario.
    - Un **`QLabel`** que diga "Ingresa tu nombre de usuario".
    - Un **`QLineEdit`** para ingresar la contraseña. Esta debe ocultar la contraseña para resguardar la privacidad de los usuarios de DCCubitos. Para esto, debes utilizar el método

setEchoMode que recibe como argumento exacto **QLineEdit.Password**.

- Un **QLabel** que diga "Ingresa tu contraseña".
- Un **QPushButton** para enviar información de *login* y poder iniciar el juego. Debes conectar la señal **clicked** de este con el método **enviar_login**.

- Modificar `def enviar_login(self) -> None`
  Este método emite la señal para que se efectúe la verificación de los datos de *login* en el *back-end*. Debes utilizar la señal **self.senal_enviar_login** para enviar el nombre de usuario y la contraseña ingresada.

- Modificar `def recibir_validación(self, valid: bool, errores: set) -> None`
  En este método debes recibir el resultado de la validación realizada en el método `comprobar_usuario` de la clase de *backend* `LogicaInicio`, y en base a este ejecutar las acciones correspondientes. Recibe como argumentos un `bool` que indica si la validación fue exitosa y un `set` que indica si ocurrieron errores. Este último contiene el string `"usuario"` si hay un error en el nombre de usuario, y el string `"contraseña"` si hay un error en la contraseña (o ambas si hay error en ambas cosas).
  - Si los datos ingresados fueron correctos debes esconder esta ventana.
  - Si los datos tuvieron un error, debes notificarlo en algún espacio adecuado de la ventana según el error (ya sea con el mensaje `"usuario inválido"` o `"contraseña inválida"`) y **no** esconder la ventana.

  HINT: Para notificar los errores puedes usar el método `setPlaceholderText("texto")` de los elementos `QLineEdit`. Si usas esto, antes **debes** limpiar el *input* con el método `.setText("")`

Métodos de back-end:

La lógica de la ventana de inicio se encuentra en la clase **LogicaInicio**. El archivo en donde deberás trabajar para esa parte es `backend/logica_inicio.py`.

- **Métodos ya implementados**
  - No modificar `def __init__(self) -> None`:
    Inicializa la clase **LogicaInicio**
- **Métodos que deberás implementar**
  - Modificar `def comprobar_usuario(self, usuario: str, contrasena: str) -> None`:
    Este método debe comprobar si los datos ingresados en la página de inicio son correctos. Para validar, debes revisar que:
    - El nombre de usuario sea **alfanumérico** (es decir tiene sólo letras y/o números).
    - La contraseña **no** esté dentro de las `CONTRASENAS_PROHIBIDAS` que se encuentran en el archivo `parametros.py`.

    Luego, sólo si la validación fue exitosa, es decir, el usuario y contraseña cumplen las condiciones solicitadas, debes emitir la señal: `senal_abrir_juego` con el nombre de usuario como parámetro.
    Finalmente, debes emitir la señal `senal_respuesta_validacion` que envía un *bool* con el resultado de la validación y un *set* con todos los errores posibles. Este *set* será vacío si no hubo ningún error, tendrá el *string* `"usuario"` si el nombre de usuario es inválido, y/o el *string* `"contraseña"` si la contraseña es inválida.

Señales:

Deberás conectar las señales con los respectivos métodos en el archivo `dccubitos.py`, en el método `conectar_inicio` de la clase `DCCubitos`.

- **Señales de VentanaInicio**
  - Modificar `senal_enviar_login`: Esta señal envía el nombre de usuario y contraseña ingresados. Debes conectarla con el método **`comprobar_usuario`** de la clase **`LogicaInicio`**.
- **Señales de LogicaInicio**
  - Modificar `senal_respuesta_validacion`: Esta señal envía un *bool* indicando el estado de la validación y un set con los errores (un set vacío si no hay errores). Debes conectarla con el método **`recibir_validacion`** de la clase **`VentanaInicio`**.
  - Modificar `senal_abrir_juego`: Esta señal envía un *string* con el nombre de usuario. Debes conectarla con el método **`mostrar_ventana`** de la clase **`VentanaJuego`**.

Parte 2: Ventana de juego

Luego de ingresar correctamente los datos en la ventana de inicio, se procederá a abrir la ventana de juego. En esta parte deberás hacer que se muestre correctamente la ventana de juego y completar los métodos necesarios para el funcionamiento del juego. Por último, deberás conectar y emitir las señales que permiten la comunicación entre el front-end y el back-end del juego.

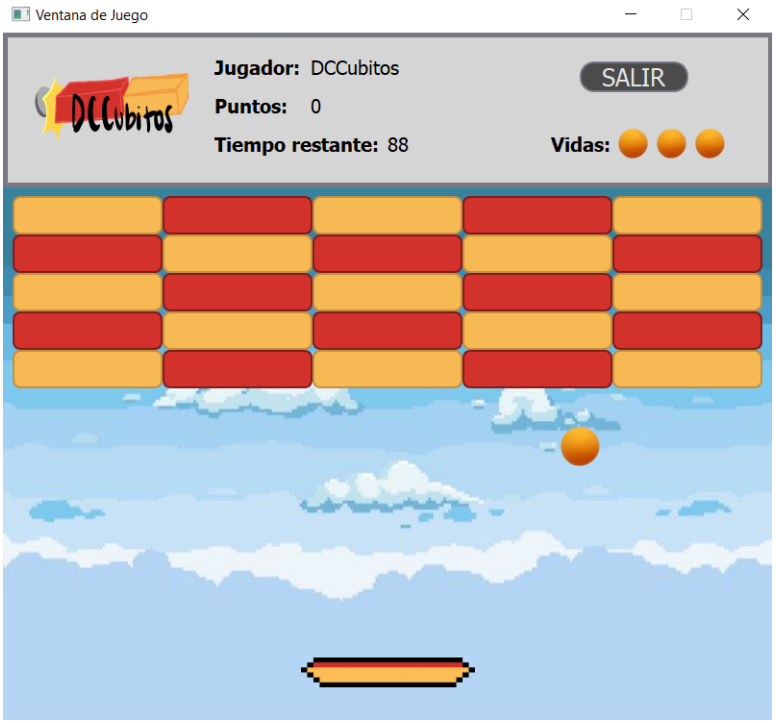

Figura 2: Vista de ventana de juego

Métodos de front-end:

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

- **Métodos ya implementados**

- No modificar `def __init__(self) -> None:`
  Inicializa la ventana de juego y llama al método `init_gui(self)`.
- No modificar `def init_gui(self) -> None:`
  Este método agrega título a la ventana, llama al método `asignar_bloques(self)`, crea una lista con los labels de las vidas y conecta el botón de salir del juego.
- No modificar `def setear_datos(self, datos: dict) -> None:`
  Este método muestra los datos del jugador en la parte superior de la ventana de juego.
- No modificar `def asignar_bloques(self) -> None:`
  Este método asigna los bloques de juego a un diccionario que los contendrá a todos.
- No modificar `def actualizar_datos(self, datos: dict) -> None:`
  Modifica los labels de **Puntaje** y **Tiempo** según va avanzando el juego.
- No modificar `def mover_plataforma(self, posicion: tuple) -> None:`
  Se encarga de actualizar la posición del label de la plataforma.
- No modificar `def mover_pelota(self, posicion: tuple) -> None:`
  Es la encargada de modificar la posición del label de la pelota.
- No modificar `def eliminar_bloque(self, numero_bloque: int) -> None:`
  Oculta el label del bloque correspondiente según el número de bloque entregado.
- No modificar `def bajar_vida(self, vidas: int) -> None:`
  Según el número de vidas que recibe, se encarga de ocultar el ícono de las vidas.
- No modificar `def reset_labels(self) -> None:`
  Restablece todos los bloques y vidas que hayan sido ocultados en la partida.

- **Métodos que deberás implementar**
  - Modificar `def mostrar_ventana(self, usuario: str) -> None:`
    Este método debe mostrar la ventana de juego y emitir la `senal_iniciar_juego` con el usuario como parámetro.
  - Modificar `def keyPressEvent(self, event: QKeyEvent) -> None:`
    Este método se llama cada vez que se presiona una tecla. Lo que deberás hacer será emitir la `senal_tecla` con el valor (en **minúscula**) de la tecla presionada. Para obtener el valor de la tecla presionada puedes usar `event.text()`.

Métodos de back-end:

El archivo en donde deberás trabajar para esa parte es `backend/logica_juego.py`. El funcionamiento del juego se encuentra en la clase `LogicaJuego`.

- **Métodos ya implementados**
  - No modificar `def __init__(self, plataforma: Plataforma, pelota: Pelota) -> None:`
    Inicializa el juego con su respectiva plataforma y pelota, además define las vidas y el puntaje iniciales. Crea tres `QTimers` para el correcto funcionamiento del juego. Llama al método `self.crear_bloques` para construir los bloques del juego y al método `self.configurar_timers` para configurar los *timers* creados.
  - No modificar `def crear_bloques(self) -> None:`
    Crea los bloques iniciales del juego.

- No modificar `def iniciar(self, usuario) -> None`:
  Inicia los *timers* del juego y emite la señal con los datos de inicio del juego.
- No modificar `def mover_pelota(self) -> None`: Se responsabiliza del movimiento de la pelota, revisa las colisiones con la plataforma, los bloques y las paredes.
- No modificar `def revisar_ganador(self) -> bool`:
  Retorna `True` si no queda ningún bloque activo (se gana el juego). Retorna `False` en otro caso.
- No modificar `def bajar_vida(self) -> None`:
  Resta una vida y baja el puntaje respectivo a perder una vida.
- No modificar `def eliminar_bloque(self, posicion: tuple) -> bool`:
  Elimina un bloque que ha sido colisionado por la pelota.
- No modificar `def actualizar_juego(self) -> None`:
  Emite una señal para actualizar los datos en pantalla a medida que avanza el juego.
- No modificar `def terminar_juego(self) -> None`:
  Detiene los *timers* del juego y emite las señales respectivas para terminar el juego, cerrar la ventana de juego y restaurar las posiciones iniciales.
- No modificar `def reset_datos(self) -> None`:
  Restaura todos los valores del juego y deja todo como al inicio.
- No modificar `def cheatcode(self, tecla: str) -> None`:
  Termina el juego destruyendo automaticamente todos los bloques (es una pequeña ”trampa”para completar el juego con un buen puntaje).

- **Métodos que deberás implementar**
  - Modificar `def configurar_timers(self) -> None`:
    En este método debes configurar los *timers* definidos en `__init__()` para el funcionamiento del juego:
    - `self.timer_juego`: este *timer* debe tener un intervalo de `TIEMPO_JUEGO`, y su *timeout* debe estar conectado al método `self.terminar_juego`. Además debes indicar que este timer no se repite con `.setSingleShot(True)`.
    - `self.timer_actualizar_juego`: este *timer* debe tener un intervalo de `ACTUALIZAR_JUEGO` y su *timeout* debe estar conectado al método `self.actualizar_juego`.
    - `self.timer_pelota`: este *timer* debe tener un intervalo de `ACTUALIZAR_PELOTA` y su *timeout* debe estar conectado al método `self.mover_pelota`.
  - Modificar `def mover_plataforma(self, tecla) -> None`:
    Si la tecla recibida es `TECLA_IZQUIERDA` o `TECLA_DERECHA` debes modificar la posición de la plataforma utilizando el método `mover` de la clase `Plataforma` y luego emitir `senal_mover_plataforma` con la nueva posición de la plataforma. En el caso que la tecla corresponda a `TECLA_CHEATCODE_KO` deberás utilizar el método `self.cheatcode()`.

Señales:

Deberás conectar las señales con los respectivos métodos en el archivo `dccubitos.py`, en el método `conectar_juego` de la clase `DCCubitos`.

- **Señales de `VentanaJuego`**

- Modificar `senal_iniciar_juego`: Esta señal inicia el funcionamiento del juego. Debes conectarla con el método `iniciar` de la clase `LogicaJuego`
- Modificar `senal_tecla`: Esta señal envía un *string* con la dirección de movimiento de la plataforma. Debes conectarla con el método `mover_plataforma` de la clase `LogicaJuego`

- **Señales de `LogicaJuego`**
  - Modificar `senal_mover_plataforma`: Esta es la señal para mover la plataforma en la ventana de juego, envía una *tuple* con su posición. Debes conectarla con el método `mover_plataforma` de la clase `VentanaJuego`.
  - No modificar `senal_cargar_datos_iniciales`: Esta señal envía los datos del juego en un *dict* para que se muestren en pantalla. Debes conectarla con el método `setear_datos` de la clase `VentanaJuego`.
  - No modificar `senal_mover_pelota`: Esta es la señal para actualizar la posición de la pelota en la ventana de juego, envía una *tuple* con la posición de la pelota. Debes conectarla con el método `mover_pelota` de la clase `VentanaJuego`.
  - No modificar `senal_enviar_datos`: Esta es la señal para actualizar los datos en pantalla (ej: los puntos), envía un *dict* con los datos. Debes conectarla con el método `actualizar_datos` de la clase `VentanaJuego`.
  - No modificar `senal_eliminar_bloque`: Esta señal envía un *int* indicando el bloque a eliminar de la pantalla. Está conectado con el método `eliminar_bloque` de la clase `VentanaJuego`.
  - No modificar `senal_bajar_vida`: Esta señal indica que se debe bajar una vida de las mostradas en pantalla. Está conectada con el método `bajar_vida` de la clase `VentanaJuego`.
  - No modificar `senal_cerrar_ventana_juego`: Es la señal que indica que se debe cerrar la ventana de juego. Está conectada con el método `salir` de la clase `VentanaJuego`.
  - No modificar `senal_reset_ventana`: Es la señal que indica que se deben restaurar los valores iniciales de la ventana. Está conectada con el método `reset_labels` de la clase `VentanaJuego`.
  - No modificar `senal_terminar_juego`: Es la señal que indica que debe terminar el juego y debe abrirse la ventana de postjuego, envía los valores finales del juego en un *dict* para mostrarlos en la ventana de postjuego. Está conectada con el método `abrir` de la clase `VentanaPostjuego`.

Parte 3: Ventana de post-juego

Una vez se acabe el tiempo (o se termine el juego de otra forma) aparecerá una ventana con el puntaje del usuario y unos botones que le permiten volver a la ventana de inicio o salir del juego. Esto se encuentra en el archivo `ventana_postjuego.py`. No debes modificar este archivo.

Señales:

Las siguientes señales se encuentran en el método `conectar_postjuego` la clase `DCCubitos` en el archivo `dccubitos.py`.

- **Señales de `VentanaPostjuego`**
  - No modificar `senal_abrir_inicio`: Es la señal que indica que se quiere juegar de nuevo. Está conectada con el método `show` de la clase `VentanaInicio`.

Figura 3: Vista de ventana de post-juego

- No modificar `senal_cerrar_juego`: Es la señal que indica que se quiere salir del juego. Está conectada a `self.exit`.

Notas

- Para la ventana de inicio recuerda que puedes poner *layouts* dentro de otros *layouts* para que quede más ordenado. Sin embargo, también recuerda que lo importante es mostrar los elementos pedidos y no es necesario que se vea como en el ejemplo.
- Recuerda que es exigido en esta actividad que conectes todas las señales en el archivo `dccubitos.py`.
- ***Disclaimer***: DCCubitos presente un pequeño *bug* cuando la pelota colisiona simultáneamente con la parte inferior de la ventana y la plataforma. Cuando ocurre esta situación, se generar un comportamienta erratico en la pelota. **No deben preocuparse por este *bug* en el desarrollo de la actividad**. No afecta en nada lo que se busca evaluar, pero si llegaran a ocupar el código entregado para alguna otra evaluación (T2 por ejemplo), deben asegurarse de entender el código y arreglar ese *bug*.

Requerimientos

- (3.00 pts) Parte 1: Ventana de inicio
  - (2.00 pts) Completa correctamente los métodos de *front-end*:
    - (1.00 pts) Completa correctamente el método `crear_elementos`.
    - (0.50 pts) Completa correctamente el método `recibir_validacion`.
    - (0.50 pts) Completa correctamente el método `enviar_login`.
  - (0.50 pts) Completa correctamente el método `comprobar_usuario`.

- (0.50 pts) Conecta correctamente las señales

- (3.00 pts) Parte 2: Ventana de juego
  - (1.00 pts) Implementa correctamente los métodos de *front-end*:
    - (0.50 pts) Completa correctamente el método `mostrar_ventana`.
    - (0.50 pts) Completa correctamente el método `keyPressEvent`.
  - (1.50 pts) Implementa correctamente los métodos de *back-end*:
    - (0.75 pts) Completa correctamente el método `configurar_timers`.
    - (0.75 pts) Completa correctamente el método `mover_plataforma`.
  - (0.50 pts) Conecta correctamente las señales