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

14 de mayo de 2020
Actividad Formativa

Actividad Formativa 04

Interfaces Gráficas I

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AF04/`
- **Hora del *push*:** 16:50

Importante:

- **Inconvenientes durante la actividad:** En caso de que tengas algún problema con el desarrollo o entrega de tu actividad, causado por un factor fuera de tu control, envía un *email* al correo oficial del curso (iic2233@ing.uc.cl) explicando lo ocurrido. El equipo docente analizará tu caso e intentará ayudarte a encontrar una solución.
- `.gitignore`: Para esta actividad te entregamos dos carpetas con archivos necesarios para el funcionamiento de la actividad, sin embargo, para evitar subir una gran cantidad de duplicados a GitHub, antes de subir cualquier cosa, deberás crear un archivo `.gitignore` que contenga los nombres de las carpetas (`Sonidos`, `Sprites`), para así no entregar esas carpetas.
- **Git:** 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

En un mundo fantástico medieval, después de un arduo y largo trabajo de *Witcher* (Cazador de monstruos) has decidido tomar un descanso en una de las tabernas de la ciudad. Mientras disfrutabas una rica comida para recuperar energías, escuchas entre la multitud la voz de *Dahntelik Pintovrim*, el renombrado tesorero de la nación *DCCountry*, y se activan tus instintos de aventura. Usando tus altos puntos de carisma, logras convencer a *Dahntelik Pintovrim* para un duelo de **DCCuent**. Si ganas, obtendrás la preciada imagen sagrada, pero si pierdes sufrirás el peor de los destinos.

Reglas del Juego

Este juego se divide en rondas, y se enfrentan dos jugadores, el usuario (**Jugador 1**) y el programa (**Jugador 2**). En cada ronda, se les provee a los jugadores tres cartas, de las cuales deben elegir una, con la esperanza de vencer la carta elegida por el contrincante. Para determinar cuál carta elegida gana se revisan las propiedades de las cartas elegidas: su **tipo** y **valor**. Los tipos existentes son: **Infantería**, **Artillería** y **Rango**; y para determinar quien vence un enfrentamiento, se siguen las siguientes reglas:

- Infantería le gana a Rango, Rango le gana a Artillería, y Artillería le gana a Infantería.
- Si ambas cartas son del mismo tipo, gana la que tenga valor más alto.
- En el caso de que ambas tengan el mismo tipo y valor, se considera que nadie gana.

El ganador de una partida es aquel en ganar cuatro rondas separadas.

Infantería, Rango, y Artillería.

El Programa

Para ayudarte a traer a la vida este juego, se te provee el *back-end* del programa implementado, **que no debes modificar**, y también parte del esqueleto del *front-end*, **el cual debes completar**. En total son dos ventanas a completar, y deberás también conectar señales entre componentes del programa.

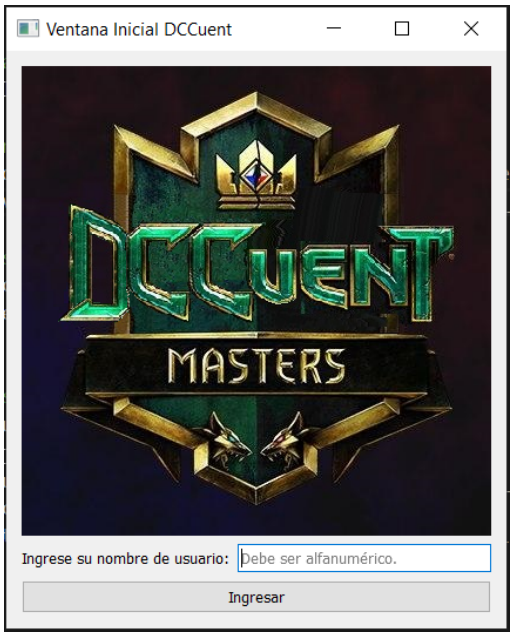
La lógica de juego (*back-end*), como elegir cartas para cada ronda, elegir cartas para el jugador enemigo, y revisar quien gana cada ronda, ya está implementado en el módulo `logica.py`, en la clase `Logica`. Se modeló de tal forma que se comunicará con el *front-end* mediante señales que se explican más adelante.

El programa tendrá una ventana de inicio, una de juego, una de enfrentamiento y finalmente una con el resultado de la partida. Se detalla las funciones de cada ventana y qué partes deberás completar en las siguientes secciones.

El archivo `main.py` es el módulo principal del programa. En este se instancia `Logica` y los esqueletos de ventanas del *front-end*. Con estas instancias creadas se efectúa la conexión de señales entre *back-end* y ventanas, algunas se entregan completadas, **mientras que otras debes conectar**.

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

Ventana inicial



La ventana inicial se encuentra incompleta y es la ventana que da la bienvenida al programa, además es donde se ingresa el nombre usuario del jugador. Al ingresarse un nombre de usuario válido (alfanumérico), se comienza el juego y se pasa a la siguiente ventana. El *front-end* de esta ventana se encuentra en la clase `VentanaInicial` del archivo `ventana_inicial.py` y se estructura de la siguiente manera:

- Métodos:
  - `def __init__(self, *args)`: Instancia la ventana. **No debes modificarlo**.
  - `def crear_pantalla(self)`: Crea y agrega el contenido gráfico a la ventana. Se encuentra (casi) vacío y debes agregarle los siguientes elementos:
    - Logo de la actividad (`QPixmap`). La ruta de la imagen se encuentra representada en la variable `ruta_logo`. Se recomienda un tamaño de 400 por 400.
    - Campo de texto (`LineEdit`) donde se pueda ingresar el nombre del jugador. Este debe ser almacenado en un atributo de instancia llamado `input_usuario`.
    - Botón (`QPushButton`) para accionar revisión de nombre e intentar ingresar al juego.

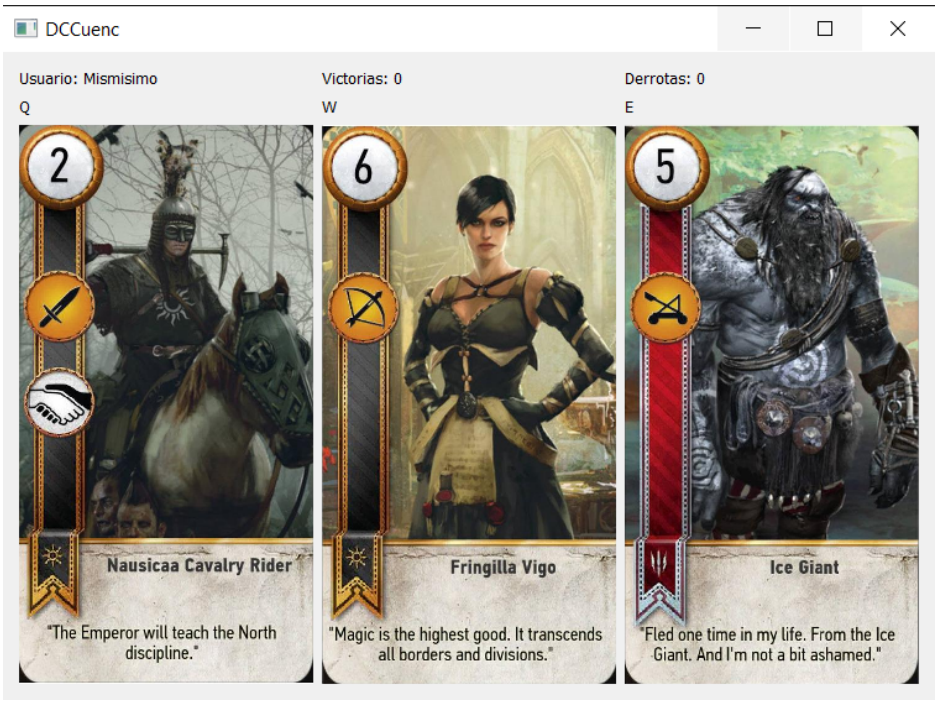
    Puedes utilizar (o no) algún tipo de *layout* para ordenar todo este contenido, puedes usar la imagen anterior como referencia.
  - `def revisar_input(self)`: Este método se encuentra vacío y se debe accionar cuando el botón de la ventana es presionado. Lo que hace es enviar el nombre ingresado hacia el *back-end*, para que sea revisado. Debes implementarlo de tal forma que se obtenga el texto escrito por el usuario, y se emita mediante la señal `senal_revisar_nombre`.
  - `def recibir_revision(self, error)`: Método que se acciona con el resultado de revisión desde el *back-end*. Actualiza la ventana luego de verificar el nombre de usuario. Este método ya está implementado y **no debes modificarlo**.

- Señales (**no debes modificarlas**):
  - `senal_revisar_nombre`: Señal encargada de comunicar desde *front-end* con la revisión de nombre en *back-end*. La conexión de la señal con el método `verificar_nombre` (de `Logica`) ya está realizada en `main.py` (línea 28) **y no debes modificarla**.
  - `senal_resultado_verificacion`: Esta señal se define en `Logica` y es la encargada de comunicar el resultado de verificación desde el *back-end* hacia el *front-end*. La conexión con el método `recibir_revision` ya está realizada en `main.py` (línea 29) **y no debes modificarla**.

Ventana de juego

Una vez validado el nombre de jugador, se abre la ventana de juego, que deberás completar. En esta se muestran tres cartas disponibles y sobre cada carta se indica qué tecla debe presionar el usuario para seleccionarla. Además, en la misma ventana se muestra el nombre de usuario, cuantas victorias lleva y cuantas derrotas lleva.

De forma similar a la ventana anterior, la clase `Logica` posee métodos listos que controlan la lógica de juego y debes conectar con esta ventana. Por otra parte, se te entregan las imágenes de cartas para que las puedas utilizar.



Lo que debes hacer es editar el *front-end*, que se encuentra en la clase `VentanaPrincipal` del archivo `ventana_principal.py`, y se estructura de la siguiente manera:

- Métodos:
  - `def __init__(self, *args)`: **No debes modificarlo**.
  - `def crear_pantalla(self)`: Se encarga de crear la base de la ventana, sin embargo no está completo. Nota que este método se llama cuando se instancia la ventana, no cuando se muestra. Entonces este método solo crea **bases de elementos sin contenido**, es en **otro método** donde se poblarán con contenido. Debes agregar elementos para:

    - Mostrar el nombre del usuario (`QLabel`).
    - Mostrar las victorias y derrotas (`QLabels`).
    - Mostrar las teclas asociadas a cada carta (`QLabels`). `Q` para jugar carta de infantería, `W` para jugar carta de Rango, y `E` para jugar carta de Artillería.
    - Mostrar las cartas de la ronda, una para Infantería, una para Rango y una para Artilleria (`QLabels` de un tamaño recomendado de 238 de ancho y 452 de alto).

  Puedes utilizar (o no) algún tipo de *layout* para ordenar todo este contenido.

- `def actualizar(self, datos)`: Se encarga de actualizar el contenido de la ventana y mostrarla. Recibe un diccionario `datos` con la información necesaria para completar el contenido de la ventana. Debes implementarla de tal forma que extraiga la información relevante y actualice el contenido de la ventana. El diccionario `datos` tendrá la siguiente estructura:
    - `datos["usuario"]` contiene el nombre del usuario ingresado (`str`).
    - `datos["victorias"]` contiene el número de victorias del jugador (`int`).
    - `datos["derrotas"]` contiene el número de derrotas del jugador (`int`).
    - `datos["infanteria"]` contiene la carta de tipo Infantería a mostrar en el espacio de carta de Infantería. El objeto es un diccionario, y tiene una llave `"ruta"` cuyo valor asociado es la ruta a la imagen de la carta. El tamaño recomendado es de 238 de ancho y 452 de alto.
    - `datos["rango"]` contiene una carta, similar al caso anterior, pero para el tipo Rango.
    - `datos["artilleria"]` similar a los casos anteriores, contiene la carta de tipo Artilleria.
- `def keyPressEvent(self, evento)`: Método que atrapa evento de presión de teclado. Debes implementarlo de tal forma que verifique que se haya apretado alguna de las opciones (`Q`, `W` o `E`), y se emita la señal al *back-end* con la carta escogida mediante la señal `senal_enviar_jugada` (debe enviar el objeto diccionario asociado). Finalmente se debe esconder la ventana después de emitir la información.

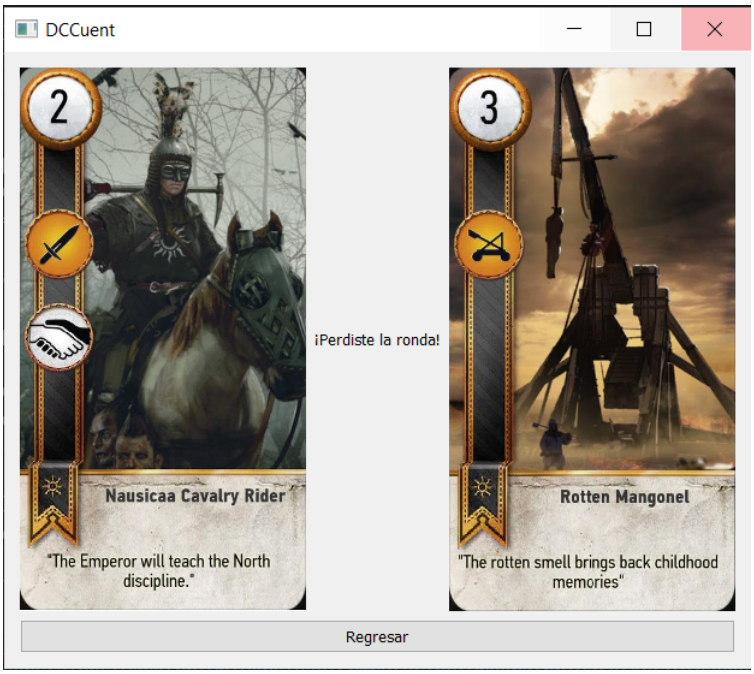
- Señales (**hay una que debes conectar**):
    - `senal_comenzar_juego`: Esta señal se define en `Logica` y es la encargada de abrir y comenzar la primera ronda del juego en el *front-end*. La conexión con el método `actualizar` de la ventana de juego ya está realizada en `main.py` (línea 37) **y no debes modificarla**.
    - `senal_enviar_jugada`: Señal encargada de enviar desde *front-end* la carta jugada hacia el *back-end*, para que este procese el resultado de la ronda. La conexión entre métodos no está realizada, por lo que **debes conectar** la señal con el método `jugar_carta` de la instancia de `Logica` en `main.py`.

Ventana de combate

Una vez elegida la carta en la ventana de juego, se mostrará la ventana de combate. Esta mostrará el resultado del enfrentamiento contra el enemigo. Mediante un botón "Regresar", se puede volver a la ventana de juego, o a la ventana de fin de juego en caso de que haya terminado la partida. **No debes implementar nada en esta ventana**, ya que se entrega lista en `VentanaCombate` (`ventana_principal.py`), pero **si debes realizar las conexiones de señales necesarias** para completar su funcionamiento:

- `senal_enviar_resultado_ronda`: Esta señal se define en `Logica` y es la encargada de enviar el resultado de una ronda desde *back-end* hacia el *front-end*. Lo hace emitiendo un objeto diccionario con todos los datos de resultados y puntajes actualizados. Debes conectar esta señal con el método `mostrar_resultado_ronda` de la instancia de `VentanaCombate` en `main.py`. Este método se encargará de mostrar los resultados en pantalla.
- `senal_regresar`: Esta señal se define en `VentanaCombate` y es la encargada de regresar a la ventana de juego principal. Debes conectarla con el método `actualizar` de la instancia de `VentanaPrincipal` en `main.py`.

La ventana de combate se verá así:



Ventana de fin del juego

El juego terminara una vez un jugador gane cuatro rondas y abrirá la ventana final con el resultado. Similar a la ventana de combate, el *front-end* de esta ventana está listo en `VentanaFinal` del archivo `ventana_final.py`, y **no debes editarlo**. **Solo debes conectar una última señal**:

- `senal_abrir_ventana_final`: Esta señal se define en `VentanaCombate` y es la encargada de abrir la ventana final con el resultado de la partida completa. Debes conectarla al método `crear_pantalla` de la instancia de `VentanaFinal` en `main.py`.

P.S: Esta es la parte más entretenida de la actividad, disfrutaran mucho ver esta parte completada. Se prometen GIFs y sonidos.

P.P.S: Subimos el mazo completo al siguiente respositorio: `https://github.com/SugarFreeManatee/MazoGwent`**, para que puedas probar con todas las cartas, si quieres.**

Objetivos de la actividad

Como podrás notar, reemplazamos la sección "Requerimientos" por esta sección, con la intención de señalar lo que buscamos que aprendas al realizar esta actividad. También queremos entregarte un acercamiento más directo a los criterios que se utilizarán para la asignación de puntaje de la actividad.

- Crear una ventana, con sus respectivos *widgets*, de acuerdo a un conjunto de especificaciones dado. Específicamente, completar: `VentanaInicial` y `VentanaPrincipal`.
- Hacer la conexión de un *front-end* hecho por el estudiante con un *back-end* previamente programado, a través del uso de señales. Específicamente, conectar las señales: `senal_enviar_jugada`, `senal_enviar_resultado_ronda`, `senal_regresar` y `senal_abrir_ventana_final`.