

Programación Avanzada

IIC2233 2024-2

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Lucas Van Sint Jan - Francisca Cattán



Experiencia 5

Interfaces Gráficas II

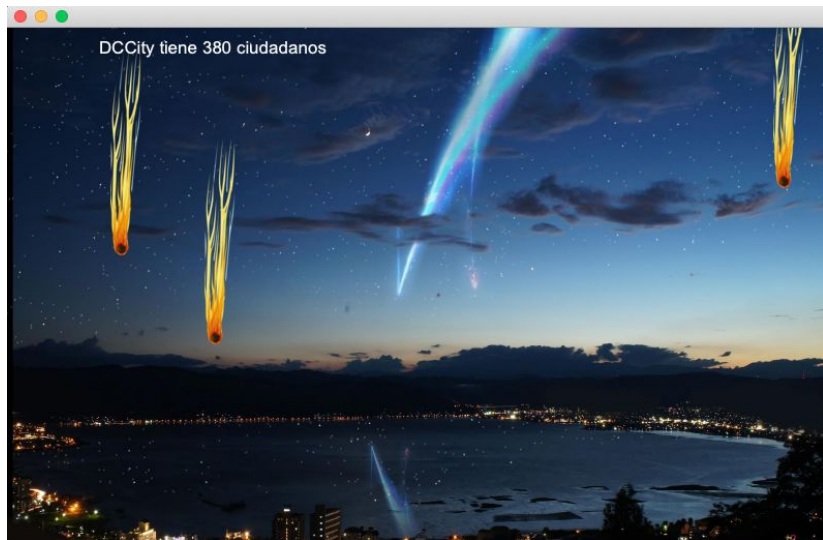
Arreglando un mini juego conectado a
un servidor

Experiencia 5: ¿Qué vamos a hacer?

1. Entender las reglas del juego junto con entender las diferentes partes del código.
2. Arreglaremos el código que presenta 11 errores que no permiten jugarlo correctamente.

Advertencia: Estos errores son algunos típicos que ocurren en la tarea de Interfaz Gráfica 😊.

Salvemos DCCity - Contexto



Se solicitó diseñar un pequeño juego, pero la persona encargada de hacerlo programó todo sin ejecutar nunca su código y luego, cuando quedaban segundos para hacer *push*, se dió cuenta que el cliente no funciona.

Ahora nos subcontrata para arreglar todo el código del cliente.

Salvemos DCCity - Parte 1



Antes de arreglar el código, vamos a entender las reglas del juego y la estructura del código.

Salvemos DCCity - Reglas

1. Al principio se solicita seleccionar una dificultad del juego, presionando ENTER u oprimiendo un botón de la ventana se enviará dicha información.
2. Una vez enviada la información el servidor nos indicará que comienza el juego y nos enviará periódicamente donde debe caer un meteorito.
3. Comenzará a sonar una canción de fondo y aparecerá nuestra ventana de juego.
4. Con el *mouse* le puedes disparar al centro del meteorito para destruirlo.
5. Si el meteorito llega a la ciudad (borde de abajo), se avisará al servidor para que nos indique cuantos ciudadanos quedan en DCCity.

Salvemos DCCity - *frontend* (Cliente)

VentanaInicio (QWidget)

- + senal_seleccionar_dificultad: pyqtSignal
- + selector_dificultad: QComboBox
- + boton_ingresar: QPushButton

- + enviar_info(): None
- + keyPressEvent(evento): None

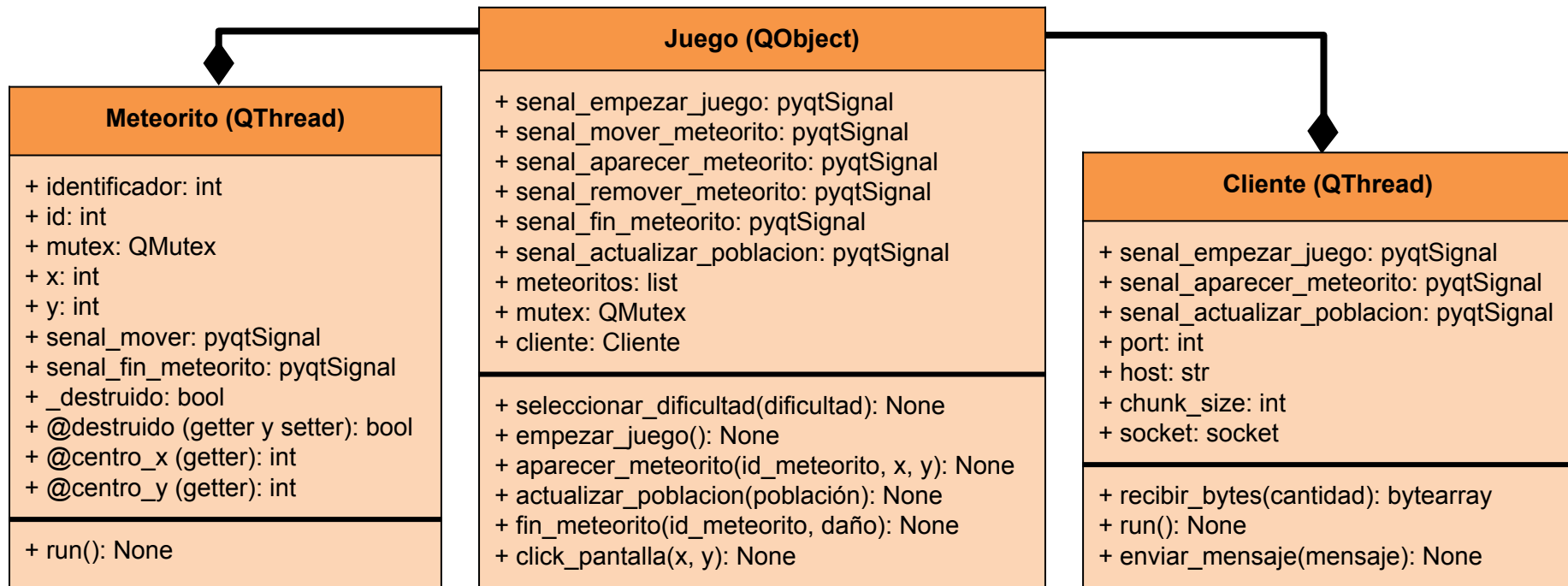
VentanaJuego (QWidget)

- + senal_click_pantalla: pyqtSignal
- + background: QLabel
- + label_vida: QLabel
- + labels_meteorito: dict
- + pixmap_meteorito: QPixmap
- + label_disparo: QLabel
- + media_player_mp3: QMediaPlayer

- + personalizarMouse(): None
- + empezar_juego(): None
- + mousePressEvent(evento): None
- + mouseMoveEvent(evento): None
- + actualizar_poblacion(texto): None
- + aparecer_meteorito(id_meteorito, x, y): None
- + remover_meteorito(id_meteorito): None
- + mover_meteorito(id_meteorito, x, y): None

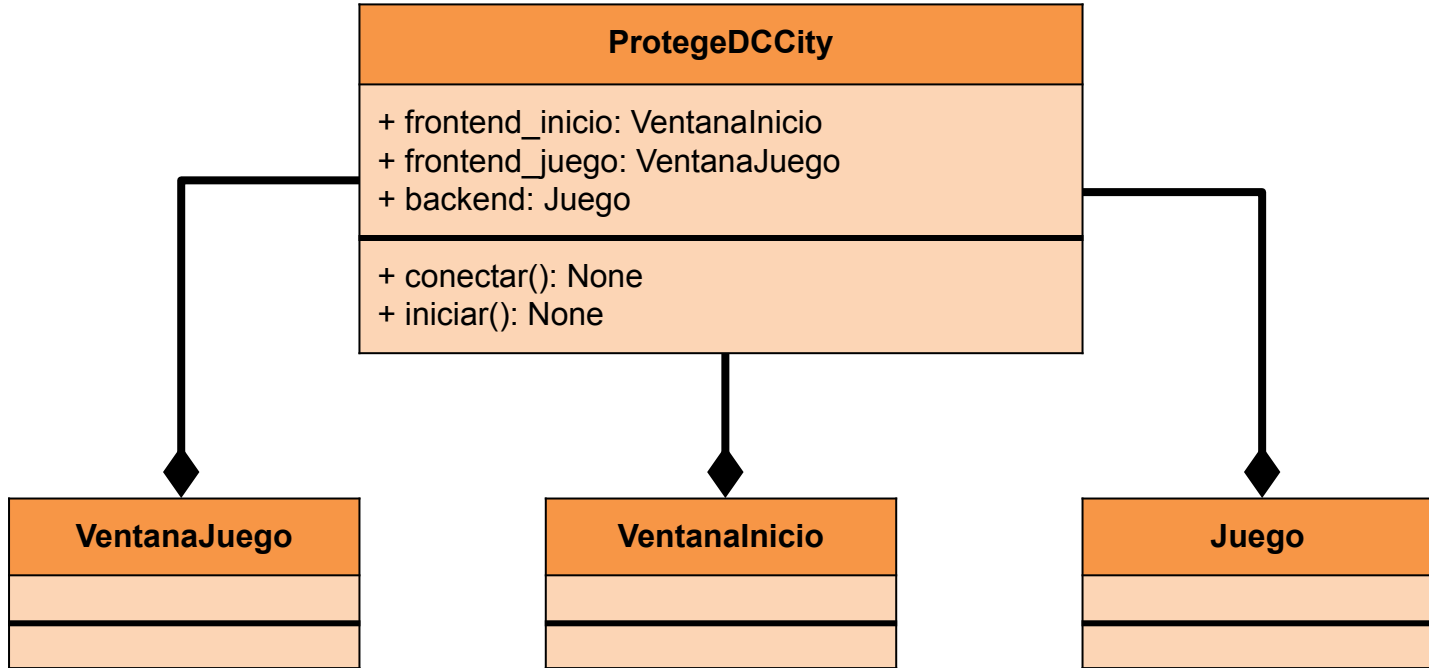
* Por temas de espacio la herencia se indicó entre paréntesis, pero debería ser una flecha.

Salvemos DCCity - *backend* (Cliente)

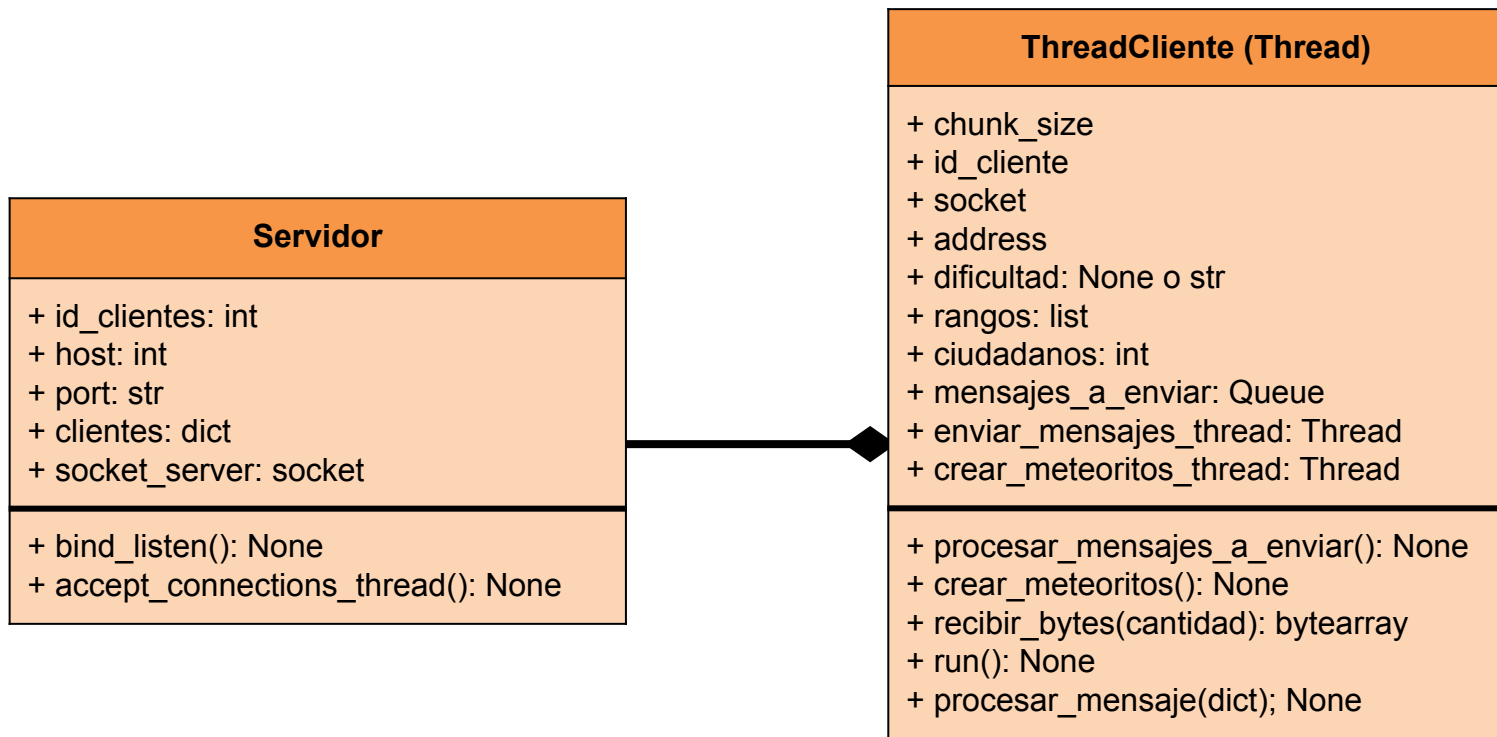


* Por temas de espacio la herencia se indicó entre paréntesis, pero debería ser una flecha.

Salvemos DCCity - *main* (Cliente)



Salvemos DCCity - *backend* (Servidor)



* Por temas de espacio la herencia se indicó entre paréntesis, pero debería ser una flecha.

Salvemos DCCity - Código

Servidor - Acotación importante

- Cuando un cliente juega, el servidor podría enviar 2 tipos de mensajes simultáneamente: la posición para poner un meteorito y la población actual en DCCity.
- Para evitar problemas de que 2 mensajes se envíen al mismo tiempo por el socket, se crea un sistema de cola (Queue) donde se almacenan todos los posibles mensajes a guardar y se crea un *thread* exclusivo para enviar los mensajes de esa cola.

Salvemos DCCity - Parte 2

 Ahora vamos a arreglar el código .

- El servidor está 100% funcional, así que se puede dejar ejecutando durante toda la experiencia. Solo vamos a arreglar el cliente.
- Se definieron “**rondas de errores**”.
 - En cada ronda se indicará cuál es el resultado esperado a llegar tras corregir los errores.
 - Luego se describirán 2 o 3 errores en el PPT que intentaremos entender cómo se gatilló y solucionarlo. También se indicará el archivo que genera ese error.
- Al final del día se publicará la solución donde se detalla cada error en el código y cómo se solucionó.

Salvemos DCCity - Parte 2

Para ejecutar el **servidor**, deberás ir a la carpeta 'Servidor' y ejecutar el siguiente comando:

```
python3 servidor.py [PUERTO*] [DIRECCION*]
```

Para ejecutar un **cliente**, deberás en otra terminal, ir a la carpeta 'Cliente' y ejecutar el siguiente comando:

```
python3 main.py [PUERTO*] [DIRECCION*]
```

* Ambas variables son opcionales. Si entregamos solo el puerto, se tomará **localhost** como dirección por defecto. Si no entregamos ninguna, se tomará **localhost** como dirección y **4444** como puerto por defecto.

Salvemos DCCity - Parte 2 (Ronda 1)

Resultado esperado tras corregir errores

- Se abra la ventana de Inicio sin ningún error en el proceso 🎉.

Salvemos DCCity - Parte 2 (Ronda 1)

1. **[networking.py]** El código se cae de inmediato con el siguiente error:

`TypeError: Cliente cannot be converted to PyQt5.QtCore.QObject in this context`

2. **[main.py]** Tras corregir el error 1, el código se vuelve a caer con el siguiente error:

`TypeError: native Qt signal is not callable`

3. **[main.py]** Finalmente no hay error, pero nunca se muestra la ventana de inicio 🙄. Hay que cerrar el proceso o abrir una nueva terminal para poder volver a ejecutar



¡A programar/arreglar!  

Salvemos DCCity - Parte 2 (Ronda 2)

Resultado esperado tras corregir errores

- Con ENTER o presionando el botón, se envíe la información al *backend*.
- En la terminal debe aparecer, para ambos casos (botón o ENTER), el mensaje:
`[Front] Empezar juego`
- Si bien aparecerá un error después de imprimir "`[Front] Empezar juego`", no nos preocupamos de eso.

Salvemos DCCity - Parte 2 (Ronda 2)

1. **[main.py]** Cuando se oprime ENTER, el *backend* nunca da aviso que recibió la información.
2. **[networking.py]** Ahora aparece un *print* de que llegó un mensaje del servidor, pero ocurre el siguiente error

TypeError: native Qt signal is not callable

3. **[ventanas.py]** Si intento oprimir el botón de enviar información de la ventana de Inicio, no se manda la información al *backend*.

¡A programar/arreglar!  

Salvemos DCCity - Parte 2 (Ronda 3)

Resultado esperado tras corregir errores

- Aparezca la ventana de Juego y reproduzca música.
- Que no aparezca un error sobre QThread en la terminal.
- Ver un meteorito caer.

Salvemos DCCity - Parte 2 (Ronda 3)

1. **[ventanas.py]** Se indica que empieza el juego, pero aparece un error en la terminal

AttributeError: 'QMediaPlayer' object has no attribute 'start'

2. **[juego.py]** Cuando se logra ver la ventana, no aparece nunca un meteorito que notifique que comienza su caída (debería salir un *print* al menos).
3. **[juego.py]** Solucionado el error 2, intentamos ejecutar y se cae el juego

QThread: Destroyed while thread is still running

¡A programar/arreglar!  

Salvemos DCCity - Parte 2 (Ronda 4)

Resultado esperado tras corregir errores

- Aparezcan muchos meteoritos y los podamos destruir cuando presionamos en el centro de ellos.
- Desaparezcan los meteoritos correctamente.
- En resumen: podemos jugar.

Salvemos DCCity - Parte 2 (Ronda 4)

Ahora saldrán errores de “jugabilidad” que no cierran el juego, pero no nos permiten jugar correctamente.

1. **[juego.py]** Aparece un meteorito que nunca desaparece 🤔
2. **[meteorito.py]** Solucionado el error 1, por fin veo muchos meteoritos, pero desde que llega el primer meteorito al final. ya no desaparecen los siguientes 😱

¡A programar/arreglar! 💻 🔧

Experiencia 3

Interfaces Gráficas II

Cierre de la experiencia

Salvemos DCCity - Resumen de errores

- Que una clase que no es del ecosistema de PyQt intente tener señales como atributo de clase.
- No hacer `show` de alguna ventana y quedarse sin nada en la pantalla.
- Usar incorrectamente una señal (no usar `.connect` o no usar `.emit`).
- Olvidar conectar una de las tantas señales que se utilizan.
- No guardar en memoria cualquier objeto de PyQt que fue creado en un método.
- No liberar el lock (`QMutex`).

Salvemos DCCity - Desafíos importante

A continuación se detallan 2 desafíos que puede que sean muy relevantes pensar en cómo implementarlos:

1. Actualmente escondemos los meteoritos que queremos "eliminar", pero eso lentamente llenará nuestra RAM. ¿existirá un modo de eliminar realmente todo lo relacionado al meteorito (QLabel y el QThread) ?
2. Actualmente, cuando el cliente va a mandar un mensaje, el *backend* se dedica exclusivamente a eso, lo cual funciona porque los mensajes son muy cortos.... pero ¿qué sucede si el mensaje es largo y el *backend* se demora mucho en enviar? ¿qué pasará con las otras acciones que debería procesar el *backend*?

Salvemos DCCity - Desafíos extras

- Agregar un contador para saber cuánto tiempo logramos sobrevivir.
- Actualmente no se gana, solo se sobrevive la mayor cantidad de tiempo.... intenta agregar un “Tiempo máximo” que permita decir que ganamos en caso de sobrevivir hasta dicho tiempo.
- Cuando empiece el juego (post seleccionar dificultad), cerrar la ventana de Inicio.
- Implementar una tecla que permita pausar/detener la música mientras jugamos.
- Implementar una tecla que permita pausar y despausar el juego.
- Implementar una tecla que permita aumentar o reducir la velocidad de caída del meteorito.

Programación Avanzada

IIC2233 2024-2

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Lucas Van Sint Jan - Francisca Cattán

