Departamento de Ingeniería Eléctrica
Facultad de Ingeniería
Universidad de Concepción
Concepción, Chile.

Mini proyecto 2:
Recreando el juego Battleship

Informe

Profesor: Vincenzo Caro F.

Integrantes: Rodrigo Hernández A.

Javier Herrera I.

2 de Junio de 2024

Taller de aplicación TIC II
Ingeniería Civil en Telecomunicaciones

Resumen

El “Mini proyecto 2” propone resolver 2 actividades distintas utilizando un kit de sensores junto con un Arduino UNO. Solo se realizó la actividad 2 está siendo ”Battleship” la cual consiste en implementar el juego Battleship donde 2 jugadores compiten en hundir barcos del oponente antes de que los suyos sean destruidos. Battleship es un juego de mesa creado en 1931 y un juego que podemos probar en web con una simple búsqueda en Google, el juego se basa en que el jugador posee 7 barcos en el cual se separan 3 tipos de barcos con diferente cantidad de vidas, estos son distribuidos en una cuadrícula donde se identifican con la letra de la columna y número de la fila que corresponde con cada barco, cada jugador organiza sus barcos a su gusto y otro jugador dirige los ataques.

Índice

1. Desarrollo

1.1. Battleship

Para la presente actividad se solicita crear un juego *Battleship* (Batalla naval), donde dos jugadores compiten para hundir los barcos del oponente antes de que los suyos sean destruidos.

El juego se distribuye en cuatro cuadrículas, dos para cada jugador, donde cada elemento individual se identifica con la letra de su columna y el número de su fila correspondiente. En la primera cuadrícula, el jugador organiza los barcos y registra los disparos del oponente, mientras que, en la segunda cuadrícula, el jugador dirige los ataques a su oponente.

Utilizando el software **Designer** que se obtiene al instalar la biblioteca **PyQt6** de Python se diseñó la interfaz gráfica con los siguientes elementos:

- 25 **Qlabels** ubicados en una matriz de 5x5 para la ubicación de los barcos.
- 25 **QPushButtons** ubicados en una matriz de 5x5 para enviar un ataque.
- 3 **QPushButtons** ubicados en una matriz 3x1 donde están las opciones de **Orden aleatorio**, **Jugar** y **Explosión de material**.
- 2 **Qlabels** para el número de rondas.
- 1 **Qlabel** para indicar el turno del jugador.
- 1 **Qlabel** para la imagen del fondo.

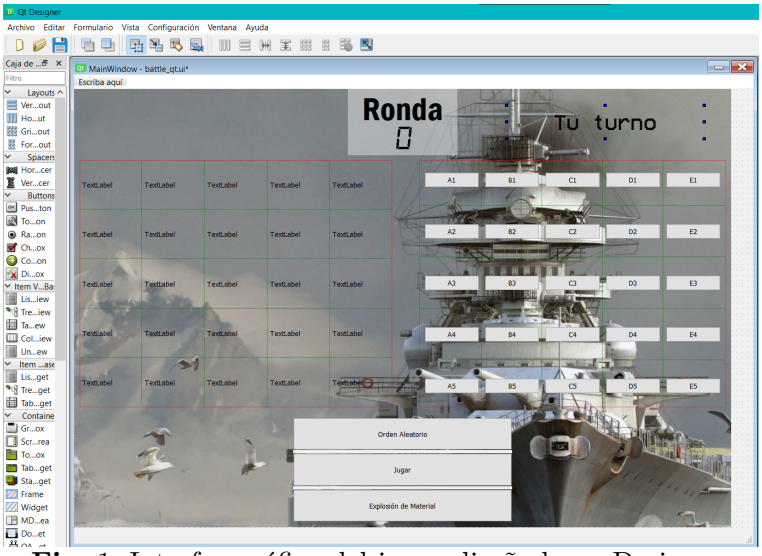


Fig. 1: Interfaz gráfica del juego diseñada en Designer.

Se crean los tipos de barco estos siendo:

- **Portaviones** que posee 3 vidas.
- **Destroyer** que posee 2 vidas.

- **Submarino** que posee 1 vida.

Todas las imágenes de los barcos fueron generadas por inteligencia artificial.

Para poder programar la interfaz gráfica se usó **Anaconda Prompt** para transformar el archivo.ui a un archivo .py usando el comando **pyuic6 -x battle_qt.ui -o battle_qt.py** y llamando al archivo desde python con el comando **from battle_qt import Ui_MainWindow**

1.2. Gameplay

El juego comenzará con el posicionamiento de los barcos de forma aleatoria y comienza cuando ambos jugadores presionan "Iniciar". Se decide de forma aleatoria qué jugador inicia y se avanzará en rondas. En las rondas cada jugador tendrá 1 turno para atacar una casilla del jugador contrincante y dependiendo del tipo de ataque y si la casilla se encuentra ocupada se actualizará la interfaz.

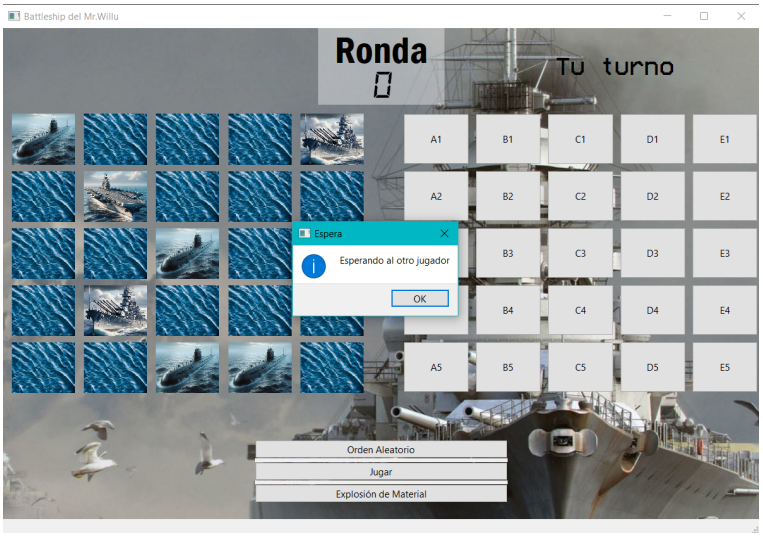


Fig. 2: Parametros iniciales del juego.

Para confirmar que ambos jugadores están listos, se creó un código en Python el cual envía un caracter del tipo **"JJ"** y espera a recibir el mismo caracter.

```python
def jugar_partida(self):
        # Mostrar alerta
        msg_box = QtWidgets.QMessageBox(self)
        msg_box.setIcon(QtWidgets.QMessageBox.Icon.Information)
        msg_box.setText("Esperando al otro jugador")
        msg_box.setWindowTitle("Espera")
        msg_box.setStandardButtons(QtWidgets.QMessageBox.StandardButton.Ok)
        msg_box.exec()

        # Enviar el carácter 'JJ' por comunicación serial
        self.serial.write(b'JJ')

        # Configurar un temporizador para verificar si se recibe 'JJ' de vuelta
        self.timer_jj = QtCore.QTimer(self)
        self.timer_jj.timeout.connect(self.verificar_jj_recibido)
        self.timer_jj.start(100)
```

Fig. 3: Función que envía JJ y espera JJ.

Una vez iniciado el juego, se permiten los siguientes ataques:

Ataque de misil

- Si se acierta y el barco es destruido, la casilla quedará bloqueada.
- Si se acierta y el barco no es destruido, la casilla actual quedará bloqueada y el barco se desplaza a una posición aleatoria, pero no a una que esté bloqueada previamente.
- Si no se acierta, el misil se hunde y no se bloquean casillas.

Explosión de material

- Un ataque que solo se puede usar una vez por jugador y se desbloqueara en la sexta ronda del juego, realiza un ataque en cruz de forma aleatoria y destruye cualquier barco sin importar su cantidad de vidas, también bloquea las casillas afectadas.

Fig. 4: Explosión de material.

- Si todos los barcos del jugador son hundidos, el jugador pierde.

1.3. Implementación

Para la implementación del juego Battleship, se diseñó una interfaz gráfica en PyQt6 la cual se observa en Figura 1, esta se encarga de mantener informado a los jugadores e ir mostrando lo sucedido en el juego (como ataques exitosos y erróneos, ganador o perdedor y el jugador con turno activo), para esto se crearon 2 cuadrículas construidas a partir de QgridLayout() estas son:

Cuadrícula principal: La cuadrícula principal muestra el estado del juego, esta se actualiza cada vez que se confirme el ataque del otro jugador. La cuadrícula posee:

- Un tablero de 5x5 compuesto por elementos de tipo QLabel().
- Imágenes personalizadas para cada tipo de barco, tanto en su condición de activo y hundido, así como también las casillas bloqueadas.

Cuadrícula de ataque: Esta cuadrícula se utiliza para ejecutar los ataques de misil, indica con cambios de colores cuando los botones que son utilizados. La cuadrícula posee:

- Tablero de 5x5 compuesto por elementos de tipo QPushButton().
- Los botones cambian a un color diferente para indicar si la casilla fue seleccionada, si hubo un impacto (verde) o si dio en el agua (naranja).

- Un botón para aleatorizar el orden de los barcos en el tablero principal (antes de iniciar el juego).
- Un botón para iniciar el juego.
- Un botón para ejecutar la explosión de material (Permanece desactivado hasta la ronda 6).

Cada vez que se acierta un ataque al enemigo, su botón correspondiente cambia a color verde. Al fallar un ataque el botón cambia a color naranja.

Cada vez que un barco tiene vida = 0, este es destruido y se actualiza su imagen con una explosión.

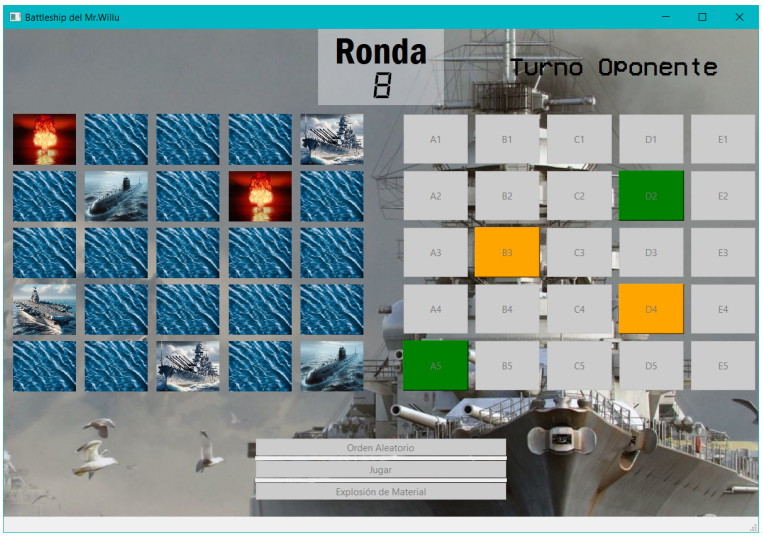


Fig. 5: Etapa avanzada del juego.

Para la comunicación del juego entre los jugadores, cada interfaz se comunica a través del protocolo serial de Arduino y ambos Arduinos están comunicados entre sí mediante I2C, esta conexión se observa en la Fig. 6.

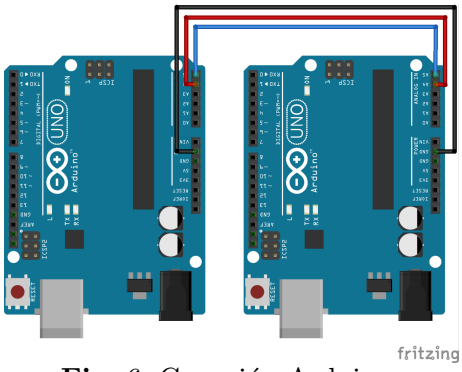


Fig. 6: Conexión Arduinos

Los Arduinos se encargan de transmitir y recibir toda información desde el inicio del juego, la conexión de los jugadores, los ataques y coordenadas, hasta el final del juego.

El Código para la comunicación I2C de los Arduinos es el siguiente:

```
#include <Wire.h>

void setup() {
  Serial.begin(9600);
```

```
  Wire.begin(8); // Inicia como esclavo en direccion 8
  Wire.onReceive(receiveEvent);
}

void receiveEvent(int howMany) {
  while (Wire.available()) { // Mientras haya datos disponibles
    char c = Wire.read(); // Lee los datos enviados por el maestro
    Serial.print(c); // Opcional: imprimir los datos para depuracion
  }
  Serial.println(); // Salto de linea despues de la lectura completa
}

void loop() {
  if (Serial.available()) {
    String str = Serial.readString();

    Wire.beginTransmission(7);  // Direccion del Arduino esclavo
    Wire.write(str.c_str());
    Wire.endTransmission();
    delay(100);  // Da tiempo para procesar
  }
}
```

2. Comentarios

La creación de la interfaz gráfica en Designer fue una tarea fácil debido a la experiencia realizada en los trabajos anteriores. Hubo un problema al correr la interfaz gráfica en Python y fue que la imagen de fondo no estaba en el fondo, esto se solucionó modificando el .py de la interfaz gráfica y comentando la linea **#self.fondo.raise_()**. También se modificó el tamaño de los botones para enviar bombas, esto debido a que en el designer no lo podía modificar. Al final se dejaron los botones de forma cuadrada.

En cuanto al código en python, con la ayuda de *ChatGPT* se creó un código base y se fue modificando agregando funciones de alerta, de lectura de datos, envío de datos y comprobación de datos. Al principio hubieron ciertos problemas debido a que se estaban leyendo datos en 2 funciones distintas al mismo tiempo. La solución fue, como se muestra en la Fig. 3, implementar un timer cuando inicia el juego y, al momento de recibir la confirmación, detener el timer e iniciar otro en otra función exclusiva que se dedica a enviar y recibir datos.

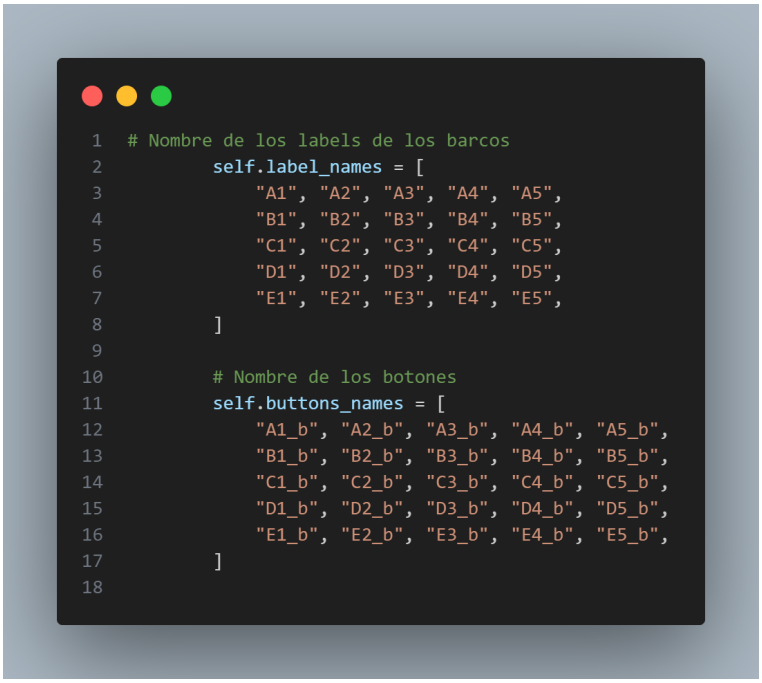
Debido a que es una función bastante grande, solo se muestra un fragmento de la función el cual lee si el dato termina en "SS" o "NN" que indica si impactó un barco o no.

```python
def procesar_confirmacion_impacto(self):
        if self.serial.in_waiting > 0:
            data = self.serial.read(self.serial.in_waiting).decode().strip()
            if data.endswith('SS'):
                posicion = data.split('SS')[0]
                self.actualizar_boton.emit(posicion)
                self.mostrar_confirmacion_impacto(posicion, True)
                self.numero_ronda += 1
                self.n_rondas.setText(str(self.numero_ronda))
                self.mi_turno = not self.mi_turno
                self.turnos_juego()

            elif data.endswith('NN'):
                posicion = data.split('NN')[0]
                self.mostrar_confirmacion_impacto(posicion, False)
                self.numero_ronda += 1
                self.n_rondas.setText(str(self.numero_ronda))
                self.mi_turno = not self.mi_turno
                self.turnos_juego()
```

Fig. 7: Función que lee y envía datos.

Las interacciones entre el código ocurren con una matriz principal donde están ubicados los barcos (Fig. 8) y exactamente el mismo nombre para los botones pero agregándole un "_b".



```
# Nombre de los labels de los barcos
        self.label_names = [
            "A1", "A2", "A3", "A4", "A5",
            "B1", "B2", "B3", "B4", "B5",
            "C1", "C2", "C3", "C4", "C5",
            "D1", "D2", "D3", "D4", "D5",
            "E1", "E2", "E3", "E4", "E5",
        ]

        # Nombre de los botones
        self.buttons_names = [
            "A1_b", "A2_b", "A3_b", "A4_b", "A5_b",
            "B1_b", "B2_b", "B3_b", "B4_b", "B5_b",
            "C1_b", "C2_b", "C3_b", "C4_b", "C5_b",
            "D1_b", "D2_b", "D3_b", "D4_b", "D5_b",
            "E1_b", "E2_b", "E3_b", "E4_b", "E5_b",
        ]

```

Fig. 8: lista de labels y botones.

La lógica del código es la siguiente:

la posición de los barcos se guarda en una variable llamada **posiciones_barcos**. Luego, el enemigo al presionar un botón (por ejemplo "A1_b"), el código elimina el "_b" y envía el dato. El contrincante lee y verifica si A1 pertenece a posiciones_barcos, de ser así envía el mismo dato junto con un "SS" (**A1SS**) el cual el enemigo lee y le indica que impactó un barco, cambiando su botón de posición a verde. una vez que se confirma si se dio o no un impacto, se cambia de ronda y pasa el turno al siguiente jugador (explicación de lo que se muestra en la Fig. 7).

Después de muchas horas de prueba y error utilizando el pc de mi novia (gracias Francisca por tu tiempo), se logró una implementación del juego completamente funcional y entretenida. Se agregaron text speech los cuales van indicando al jugador la situación en la partida y dan dinamismo a esta.

Se intentó implementar música de fondo y efectos de sonido al juego pero, por experiencias en el pasado MP1, la implementación de sonidos en python es algo complicado y en honor al tiempo esa idea se descartó (los audios quedaban en bucle y no se podían detener cuando uno quería).

Si tuviera más tiempo limpiaría el código y lo dejaría mas limpio y optimizado, esto porque al utilizar en parte a ChatGPT para el código, este creó funciones innecesarias y mucho relleno, pero de igual forma me ayudó bastante en el código.

Finalizando, estamos agradecidos con nuestro compañero de los conocimientos entregados por el profesor. Proyectos como estos ayudan mucho a pensar y divertirse programando.

3. Conclusión

Hicimos el juego un poco más difícil de lo que planeamos originalmente, lo cual fue bueno porque nos ayudó a aprender y mejorar en lo que hacemos. A medida que crecimos y aprendimos, descubrimos más sobre la comunicación I2C, una tecnología crucial que permite que los componentes electrónicos se comuniquen entre sí. Este conocimiento nos ayudó a que el hardware funcionara en conjunto de manera más fluida. Además, trabajar con interfaces gráficas nos dio la oportunidad de aprender cómo hacer que las cosas se vean interesantes y fáciles de usar. Al trabajar juntos y utilizar nuestras habilidades, mejoramos en la resolución de problemas y en la realización de las cosas.

4. Referencias

1. Colaboradores de los proyectos Wikimedia. “Batalla naval (juego) - Wikipedia, la enciclopedia libre”. Wikipedia, la enciclopedia libre. Accedido el 2 de junio de 2024. [En línea]. Disponible: https://es.wikipedia.org/wiki/Batalla_naval_(juego)