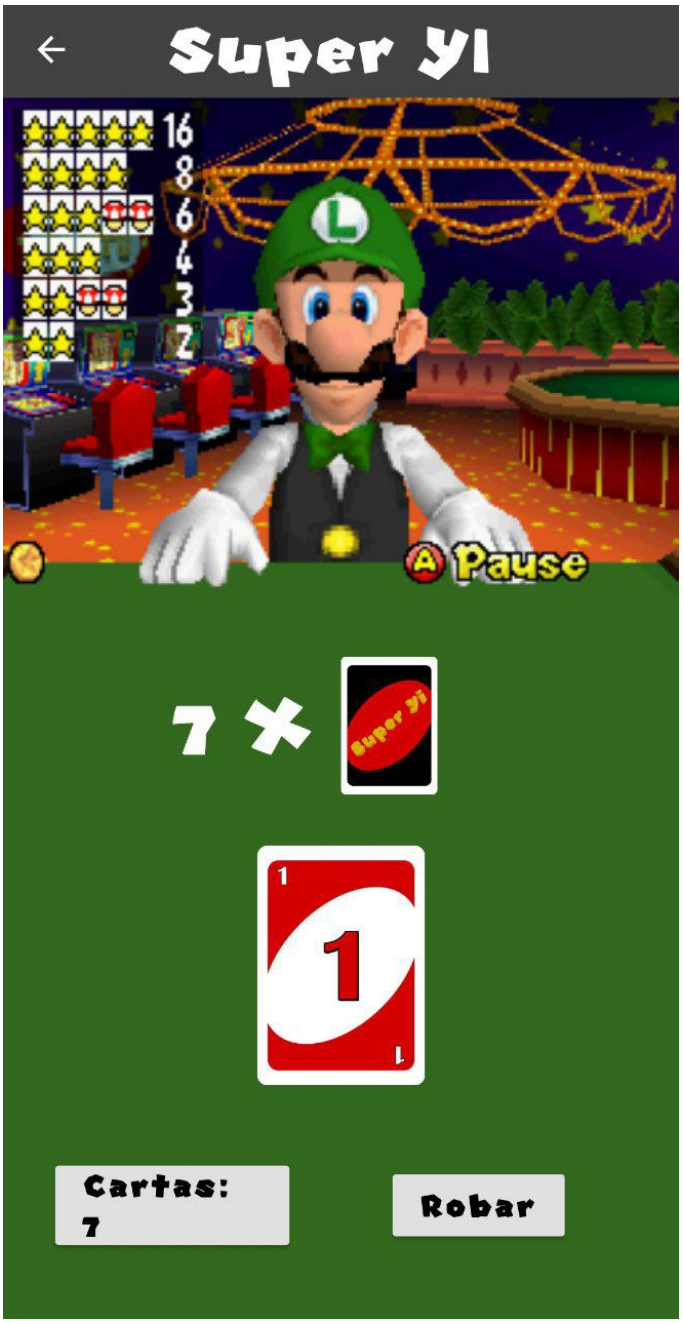
Memoria Práctica 2



Realizado por:
Antonio Quirante Hernandez
Jordi Conde Molina
Pablo Borrego Megías
Manuel Contreras Orge

Índice

Resumen de la práctica

En esta práctica hemos programado una versión bastante simple del popular juego de cartas UNO, en el que nosotros como jugador nos enfrentamos a una Inteligencia Artificial muy simple en una partida hasta que uno de los dos gane quedándose sin cartas en su mano. La práctica ha sido programada en Dart, un lenguaje de programación de Flutter.

Cada jugador contará con una mano compuesta por 7 cartas aleatorias del mazo de cartas existentes, que cuenta al principio de todo con 40 cartas (4 de cada número y 10 por color), del que, como se ha dicho anteriormente, se extraerán 7 cartas aleatorias para cada jugador más una carta más que será la que se encuentre en al tablero al principio de y sobre la que se empezará a jugar. También hay que tener en cuenta que, una vez que una carta que se encuentra en el tablero sea reemplazada por otra, esta carta volverá al mazo de cartas para poder usarse en el futuro.

La inteligencia artificial está diseñada para recorrer todas sus cartas desde la primera hasta la última para ver si tiene una carta que coincida en color o en número (en ese orden), y en caso afirmativo hecha la primera carta que coincida con la que está en el tablero, en caso contrario roba una carta random del mazo de las cartas.

En el caso del jugador persona (es decir, nosotros) cuando nos toque jugar nos aparecen 2 botones distintos, el primero en el que te aparecen las cartas que tienes en la mano y que puedes echar al tablero, cartas que puedes seleccionar, y en el caso de que sea una carta que se corresponde en color o número con la del tablero se echa, y en el caso contrario se muestra un mensaje por pantalla de que esa carta es imposible jugarla; el segundo realiza la acción de robar una carta, con la que se aumenta la cantidad de cartas en nuestra mano. Cuando un jugador (ya sea nosotros o la IA) se quede sin cartas se mostrará un mensaje de victoria por pantalla del jugador ganador y se volverá al menú principal, para que si queremos jugar otra partida con solo pulsar un botón esto se haga.

Clases utilizadas

En este apartado daremos un pequeño resumen de todas las clases utilizadas, además de mencionar su papel dentro del videojuego.

carta

Esta clase, como su nombre indica son las cartas que se utilizarán a la hora de jugar, como características tienen que pueden ser de un color de los cuatro disponibles; azul, amarillo, rojo o verde, dándoles uno de dichos colores de forma aleatoria, con un random.

Además del color, las cartas tendrán un número el cuál también se decidirá de forma aleatoria con un random, dicho números irán desde el 0 hasta el 9.

Una muestra de código sería la siguiente:

```
String color = "blanco";
  int numero = -1;

  carta(){
    var random = new Random();
    this.numero = random.nextInt(10);

    var c = random.nextInt(4)+1;
    if(c == 1){
     this.color = "azul";
    } else if(c == 2){
       this.color = "amarillo";
    } else if(c == 3){
      this.color = "rojo";
    } else if(c == 4){
      this.color = "verde";
    }
  }
```

jugador

A partir de esta clase definiremos los distintos tipos de jugadores de los cuáles se compondrá la partida, definiremos dos funciones esenciales para los jugadores, que serán las de elegir_carta(), que según el tipo de jugador, hará una cosa u otra, pero el concepto es que deje sacar una carta de la mano del jugador y la elimine de dicha mano y también la función de robar_carta() que el concepto es que puedas coger una carta de forma aleatoria del mazo de cartas, además de eliminar dicha carta del mazo para que no haya dos cartas iguales.

```
List<carta> mano = [];
  int cartas_restantes = -1;

  jugador() {
    cartas_restantes = mano.length;
  }

  carta elegir_carta(cartan) {
    return mano[0];
  }

  robar_carta(mazo) {

  }
```

jugadorIA

Como su nombre indica esta clase se dedicará al jugador controlado por el PC, no es exactamente una IA pues carece de una heurística compleja, ya que sus movimientos se basan en la aleatoriedad misma. Los métodos que tiene son los mismos métodos descritos en la clase jugador.dart.

Como he comentando anteriormente sus métodos se basan en la aleatoriedad, por tanto su método elegir_carta() el cuál devuelve una carta, podemos ver que coge la primera carta de la mano la cuál concuerde o en color o en número con la última carta usada en la partida.

Robar_carta() no tiene nada de especial, coge una carta aleatoria del mazo y elimina dicha carta del mazo.

```
class jugadorIA extends jugador {
  jugadorIA() : super();

  @override
  carta elegir_carta(cartan){
    bool fin = false;
    var cartaM = carta();
    cartaM.cartaColorSet("blanco");
    cartaM.cartaNumeroSet(-1);

    for (int i = 0; i < cartas_restantes && !fin; i++) {
      if (mano[i].color == cartan.color || mano[i].numero == cartan.numero)
{
        fin = true;
        cartaM = mano[i];
        mano.removeAt(i);
        cartas_restantes = mano.length;
      }
    }

    return cartaM;
  }


  robar_carta(mazo) {
    var nueva = carta();

    var random = new Random();
    var numero = random.nextInt(mazo.length);

    nueva = mazo[numero];
    mano.add(nueva);
    mazo.removeAt(numero);
    cartas_restantes = mano.length;
    return mazo;
  }
}
```

jugadorPersona

Esta clase se dedicará al humano el cuál jugará al videojuego, que el único trabajo que por su parte será elegir la carta deseada o robar carta, con los dos mismos métodos que llevamos describiendo de antes.

La principal diferencia es que esta vez el método elegir_carta() recibirá la carta deseada por el jugador, en lugar de la ultima carta utilizada como hacia el mismo método en jugadorIA, y entonces, una vez recibida dicha carta, se hará una búsqueda de dicha carta en la mano del jugador, es cogiéndola y eliminandola de su mano.

La validación de la carta se hará posteriormente en otra clase conectada a la parte gráfica del videojuego.

```
class jugadorPersona extends jugador {
  jugadorPersona(): super();

  @override
  carta elegir_carta(cartan){
    bool fin = false;
    var cartaM = carta();
    cartaM.cartaColorSet("blanco");
    cartaM.cartaNumeroSet(-1);

    for (int i = 0; i < cartas_restantes && !fin; i++) {
      if (mano[i].color == cartan.color && mano[i].numero == cartan.numero)
{
        fin = true;
        cartaM = mano[i];
        mano.removeAt(i);
        cartas_restantes = mano.length;
      }
    }

    return cartaM;
  }



  robar_carta(mazo) {
    var nueva = carta();

    var random = new Random();
    var numero = random.nextInt(mazo.length);

    nueva = mazo[numero];
    mano.add(nueva);
    mazo.removeAt(numero);
    cartas_restantes = mano.length;
    return mazo;
  }
}
```

home

En esta clase se realiza la interfaz de la página principal del programa, en la que se diferencian dos partes principales (o contenedores): el título y la parte principal. En el título sólo encontramos el nombre de nuestra aplicación y sus características.

```
appBar: AppBar(
title: Text(
      'Super YI',
      style: TextStyle(
            fontSize: 40,
            fontWeight: FontWeight.bold,
            letterSpacing: 2,
            color: Colors.red,
            fontFamily: 'Mario',
      ),
),
centerTitle: true,
backgroundColor: Colors.grey[800],
),
```

A continuación, se encuentra el contenedor principal, en el que destacan 2 elementos, una imagen y el botón que nos lleva a la página de juego.

```
Container(
      padding: EdgeInsets.fromLTRB(30, 100, 30, 30),
      color: Colors.lightGreen[900],
      alignment: Alignment.center,
      child: RaisedButton(
            onPressed: (){
                  Navigator.pushNamed(context, '/juego');
            },
            child: Text(
                  'Iniciar Nueva Partida',
                  style: TextStyle(
                        fontSize: 35,
                        fontWeight: FontWeight.bold,
                        letterSpacing: 2,
                        color: Colors.white,
                        fontFamily: 'Mario',
                  ),
            ),
            color: Colors.grey[800],
            shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(20)),
            padding: EdgeInsets.symmetric(horizontal: 40, vertical: 20),
      ),
),
```

Imagen de la página home:

juego

Esta es la clase que se encarga de que el juego se realice correctamente, ya que gestiona tanto todas las cartas de los jugadores y las del mazo como los turnos y las jugadas de ambos jugadores. Todo el apartado de GUI se ha realizado de la misma manera, añadiendo una columna principal y se han ido añadiendo filas. La primera fila es únicamente la foto principal, la siguiente fila indica las cartas restantes del jugador contrario, la siguiente indica mediante una imagen la carta que hay en el tablero, y por último, la última fila tiene dos botones, uno para ver tus cartas y el otro para robar cartas del mazo.

El código de la primera fila es:

```
Container(
      child: Image.asset(
            'assets/juego.png',
            scale: 0.1,
      ),
      alignment: Alignment.center,
),
```

El código de la siguiente fila es:

```
child: Text(
      '${cartasIA} X ',
      style: TextStyle(
            fontSize: 50,
            fontWeight: FontWeight.bold,
            letterSpacing: 2,
            color: Colors.white,
            fontFamily: 'Mario',
      ),
),
```

Las imágenes de las cartas se cargan en pantalla de forma dinámica. Los archivos de estas siguen la misma nomenclatura de {color}{número}.png, lo cual sirve para escribir su ruta usando los valores de los atributos del objeto carta. De esta forma, no es necesario guardar la ruta de la imagen como un atributo adicional. El código es:

```
child: Image.asset(
'assets/${this.actual.tablero1.ultimaCarta.getColor()}${this.actual.tablero1
.ultimaCarta.getNumero()}.png'
),
```

En la última fila, cabe destacar que el botón “Cartas : x” Al pulsarse pasa como parámetro a la página “Lista_cartas” un Array de cartas del Jugador. Si se ha elegido una carta válida la coloca en el centro y pasa de turno para juegue, si no se puede echar dicha carta sale un mensaje de error y no deja hacerlo. Si se roba una carta pasa también de turno. El código de los botones es muy extenso pero fácil de entender.

```
  dynamic result = await    Navigator.pushNamed(context, '/lista_cartas',
arguments: {
      'mano': actual.jugadorpersona.mano,
```

Aquí se le está pasando las cartas a la otra página. Una vez se elige una se hace un setState() para refrescar la información en pantalla. Lo mismo ocurre cuando el juagdor IA hace un movimiento.

El código del botón robar carta es el siguiente:

```
this.actual.jugadorpersona.robar_carta(this.actual.tablero1.getMazo());
```

main

En esta clase, que es la principal de la aplicación, se llama a las distintas clases principales que se usan en la aplicación, como son *home*, *juego* y *lista_cartas*, además de indicar que al iniciar la aplicación se empiece en la clase *home* y que se elimine el texto debug que aparecía arriba a la derecha por defecto.

```
void main() => runApp(MaterialApp(
  debugShowCheckedModeBanner: false,
  initialRoute: '/home',
  routes: {
    '/home': (context) => Home(),
    '/juego': (context) => Juego(),
    '/lista_cartas': (context) => lista_cartas(),
  },
));
```

lista_cartas

En esta clase se definen las cartas que tiene el jugador y que se muestran al pulsar el botón de elegir cartas del jugador en la pantalla de juego, además, se actualiza cada vez que se juega una carta o se roba una.

```
void actualizar(index) async{
      carta aux = data[index];
      await aux.color;
      await aux.numero;
      Navigator.pop(context, {
            'color': aux.color,
            'numero': aux.numero,
      });
}
```

Con este método se devuelve la carta que se selecciona que se selecciona en el botón de cartas.

```
body: ListView.builder(
      itemCount: cartasJ.length,
      itemBuilder: (context, index){
            return Card(
                  child: ListTile(
                        onTap: (){
                              updateCarta(index);
                        },
                        title: Text(' ${cartasJ[index].numero}
${cartasJ[index].color}'),
                        leading: CircleAvatar(
                              backgroundImage:
AssetImage('assets/${cartasJ[index].color}${cartasJ[index].numero}.png'),
                        ),
                  ),
            );
     },
),
```

Se muestran las cartas del array que se le pasa como parámetro y se van colocando en la lista de una en una iterando con el parámetro *cartasJ* de tipo List.

Para mostrar la imagen de la carta, se usa el mismo método que en la clase juego.

Ejemplo de cómo quedarían las cartas del jugador persona: