Práctica final

El Juego del 7

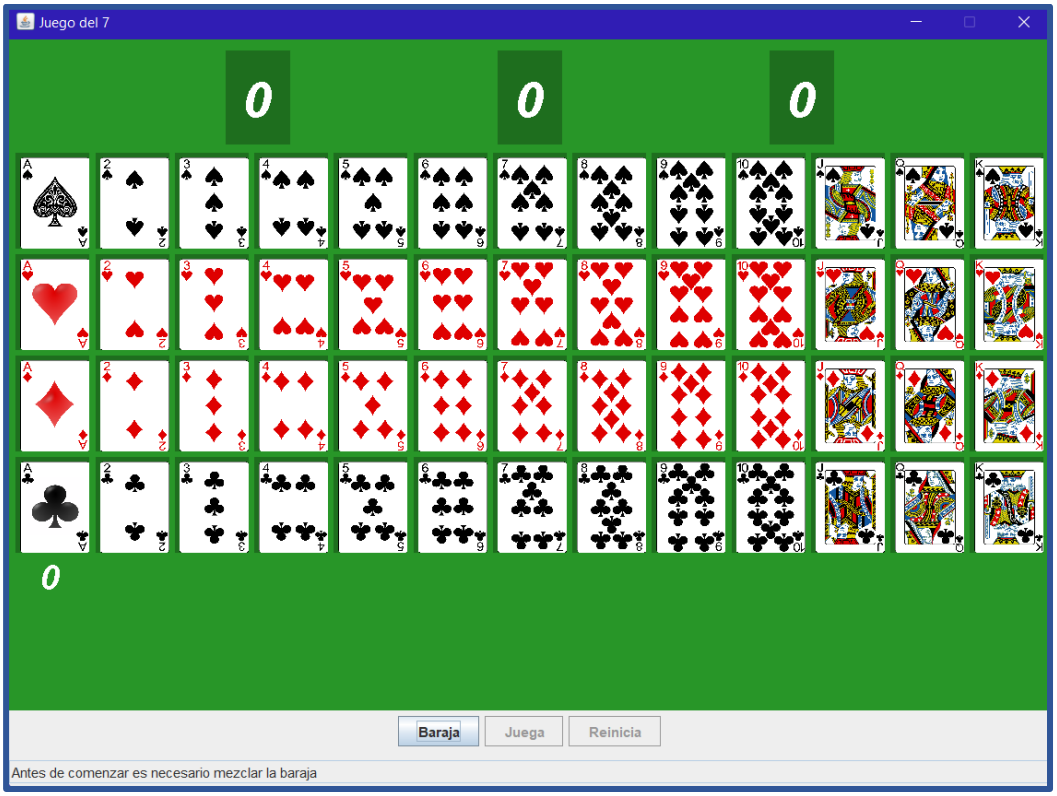
Programación II - Juan Antonio Montes de Oca Durán

Antoni Frau Gordiola

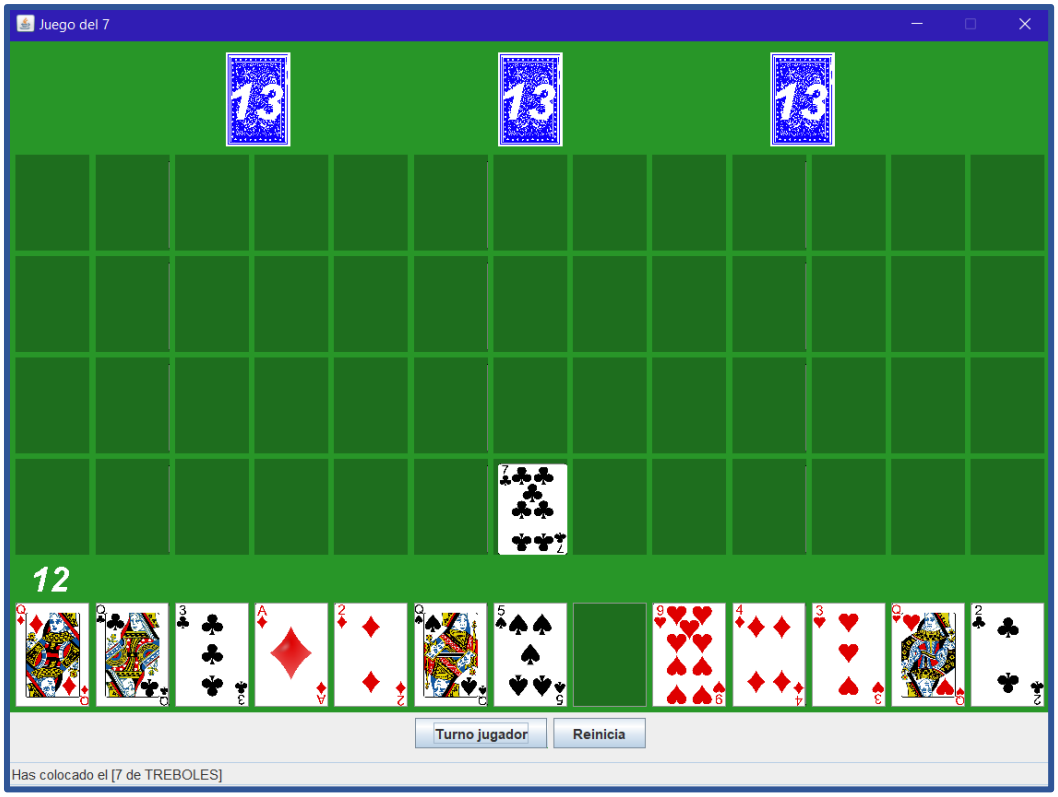
Introducción

La práctica final "El juego del 7” se trata de crear un programa en java con el entorno de desarrollo Netbeans. Esto siguiendo los estándares enseñados del paradigma de Programación Orientada a Objetos y el uso del diseño descendente aplicado a interfaces gráficas.

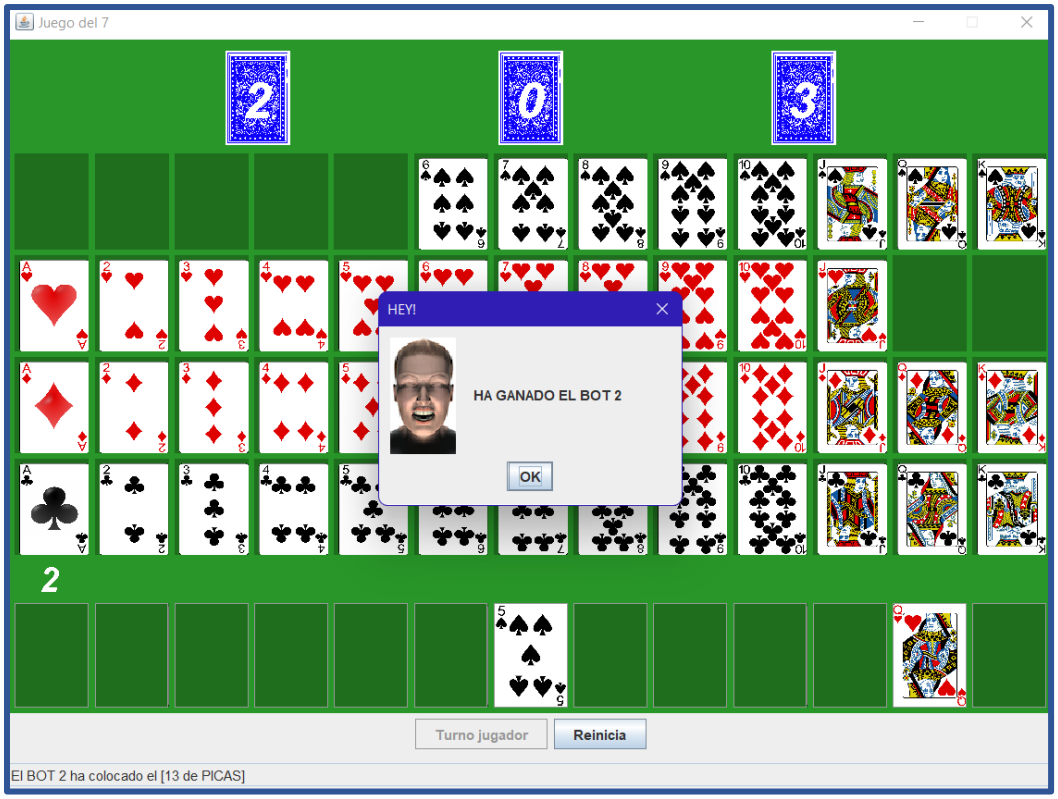
El programa debe iniciarse mostrando un tablero de juego con las cartas de la baraja francesa, indicadores para 3 jugadores de tipo máquina y uno para el jugador, además de 3 botones de acciones y un panel de texto que vaya haciendo un seguimiento de la partida. Al principio sólo debe dar la opción de barajar las cartas de la baraja, para posteriormente iniciar una nueva partida.



Al iniciar la partida con cartas barajadas con el algoritmo Fisher-Yates a cada uno de los 4 jugadores totales, el turno es del jugador 1 (el humano). En este punto el tablero se ha vacíado y preparado para poner cada jugador sus cartas, que ahora el jugador ve en la ventana, empezando por el 7 de cualquier palo.



Así seguidamente, con los sietes como cartas troncales, se deben ir poniendo una por turno de los jugadores cartas contiguas a las ya puestas: del mismo palo las cercanas por encima o por debajo del valor numérico de las cartas. Cuando, colocando cartas, llegue el momento en el que un jugador se queda sin cartas gana, y se da la opción de reiniciar, aunque esta siempre esté disponible.



Durante toda la partida se irá actualizando el campo de texto que indica las acciones del juego, así como limitar y marcar las acciones que permiten hacer los botones como pasar de turno o incluso tener en cuenta que al seleccionar en tu turno una carta no válida para jugar suene un “beep” para indicar el error.

Al pasar los turnos, cada jugador irá poniendo sus cartas en el tablero de la forma original antes de iniciar partida, de forma que alguno se quedará sin cartas y será el ganador.

Diseño

Para proporcionar el diseño final que conlleva a la estructura de la práctica, he creado una clase principal llamada ClasePrincipal en la que se llevan a cabo todos los procesos de la forma más genérica. Se completa haciendo uso de las clases Baraja, BotonCarta, Carta, Jugador, MesaJuego, PanelCarta y TextoJugador.

El vídeo explicativo del programa se encuentra en las primeras líneas de código de la clase principal y aquí.

ClasePrincipal

El desarrollo de la clase principal cuenta con el **metodoPrincipal()**, llamado desde el método ejecutor **main()**, en el que se lleva a cabo inicialización de la ventana JFrame y sus contenidos generales, así como los contenidos de estos y así sucesivamente. De esta forma, toda la gestión de interfaz gráfica y organización de los elementos en pantalla se concentra en este método. En él se crean el PanelMesa que contiene los indicadores de los jugadores máquina, el tablero y sus cartas, el indicador del jugador humano y las cartas de la mano de este. En este método también se configuran elementos como la asignación de los gestores de evento para botones espefcíficos como el de barajar o de reinicio de la partida.

Cuenta también la clase con métodos como **reiniciarPartida()** que aplica configuraciones de nueva partida en el tablero como el reinicio de las cartas de la mesa, todos los jugadores y el reincio de el texto a mostrar en el campo de texto. Otro método es **comenzarPartida()** que se encarga de repartir cartas y actualizar su visualización en pantalla, actualizar a los jugadores y sus cartas, y a otros campos gráficos como las cartas visibles que no deban serlo y el campo de texto de guía de la partida.

Un método más es **turnoBot()** que realiza, en el caso de los jugadores máquinas, las acciones de de un bot en su turno, como la elección de la carta a seleccionar y usar. El método **finalPartida()** es llamado si algún jugador gana. Este método de finalizar la partida desactiva los botones para no interactuar y muestra un mensaje de diálogo para saber quién es el ganador. **activarCartasJugador()** comprueba y activa solo las cartas que tenga asignadas el jugador, porque puede tener pero si instanciar, de forma que podrá varial la cantidad visual de cartas del jugador. De forma similar, el método **actualizarVisualizadorCartas()** obtiene las cartas de la clase MesaJuego, que se usa para gestionar las acciones y el progreso de la partida, y las aplica en las cartas de representación gráfica, lo que hace que el apartado gráfico se vaya actualizando con los cambios en las cartas.

El método **actualizarContadorJugadores()** también accede a los jugadores de la clase MesaJuego, que los gestiona, para actualizar los indicadores gráficos y mantener así actualizada la interfaz. De igual forma el método **actualizarCartasJugador()** recibe las imágenes de las cartas que tiene el propio objeto jugador en la clase MesaJuego para actualizar así las cartas gráficas de la interfaz.

Esto hace que esta clase principal se una interfaz entre el apartado gráfico y las acciones, que ocurren en las instancias de objetos MesaJuego, Jugador, Baraja, etc... Pero para que puedan interactuar con el usuario, también se requieren los gestores de eventos, existentes dentro de esta misma clase como clases privadas: GestorEventosAccion y GestorEventosCarta.

La clase GestorEventosAccion implementa la interfaz ActionListener, por lo que debe sobreescribir el métdo **actionPerformed()**. En el caso de esta clase, destinada a ejecutar una acciones dependiendo del comando del botón al que se le asigna, realiza la acción de "Jugar", "Pasa turno" y "Turno de jugador". Estas tres opciones corresponden con cargar los datos de una nueva partida, interrumpir el turno del jugador para pasar al siguiente, y realizar el turno del jugador máquina que le corresponda respectivamente.

La clase GestorEventosCarta implementa también la interfaz ActionListener y sobreescribe el método **actionPerformed()**. Esta clase está destinada a actuar al seleccionar una carta de las disponibles en el turno del jugador, por lo que podrá jugar esa carta al tablero o sonar un sonido de error si la carta no es válida. Entre otras cosas como seguir actualizando el campo de texto de guía de la partida o validar si se ha llegado al final de la partida al pasar un turno, etc.

Baraja

El motivo de esta clase es el poder aglutinar las acciones con las cartas y los conjuntos de estas. Es por eso que la clase está formada únicamente por atributos y métodos estáticos y sin constructor. A través de esta se podrán obtener la baraja estándar con las 52 cartas francesas, obtener la misma baraja pero barajada, entre otros.

Estas acciones se hacen empezando por el método **inicializarBaraja()**, que instancia objetos Carta constantes, ya que representan la baraja completa y ordenada, con sus valores numéricos, respectivos palos e imágenes desde la carpeta del proyecto. Este método se llama una sola vez al inicio de la ejecución del programa para ya tener los datos cargados.

Con el método **barajar()**, se devuelve un array de 52 Cartas barajadas con el algoritmo de Fisher-Yates, de oforma que no se pueda usar directamente en el apartado de la ventana gráfica.

El método **getPosicionDeCarta()** recibe una carta, inicializada con valores de alguna de las cartas originales y devuelve un valor entero con la posición de dicha carta respecto del conjunto de cartas originales. El método **getBaraja()** devuelve la propia barada de las 52 cartas originales. El método **getCantidadPalo()** devuelve el atributo que representa la cantidad de cartas por palo, siendo 13.

BotonCarta

Esta clase que hereda de JButton y representa las cartas que tiene el jugador para poder seleccionarlas, contiene la gráfica de los objetos carta del jugador y está destiada a ser instanciada 13 veces, ya que son las cartas que tiene el jugador en la mano.

Cuenta, por lo anterior, con un **constructor** y los métodos sobreescritos **paintComponent()** y **getPreferredSize()**. El pimero es para gestionar el apartado gráfico, que dibuja el atributo de esta misma clase que representa la carta que es por encima de un rectángulo más oscuro. El segundo método es para sobreescribir el tamaño de los botones carta a unos predefinidos, en este caso una dimensión de 56 x 82.

Cuenta también con métodos **reiniciar()**, que aplica valores iniciales al elemento gráfico, **iniciar()** para poder asignarle valores y ajustes iniciales como una imagen, y **setVisualizar()** que es usado para indicarle al objeto si se debe visualizar o no la imagen.

Carta

Esta clase, muy similar a BotonCarta pero sin el objetivo gráfico, sirve como base para el manejo de las cartas como tal, por lo que cuenta con sus atributos necesarios y métodos como su respectivo **constructor**, métodos que obtienen sus atributos como **getValor()**, **getPalo()** y **getImagen()**, y otros métodos para asignarlos como **setValor()**, **setPalo()** y 2 **setImagen()** ya que la imagen se puede asignar dada una imagen o obtener a partir del nombre de esta.

Otros métodos de objeto útiles son **copiar()**, para crear un nuevo objeto Carta idéntico o el sobreescrito **toString()** para obtener en texto una descripción del objeto.

Cuenta también con **esContigua()**, que recibe un objeto Carta y devuelve un valor booleano indicando si las dos cartas son o no contiguas por valor dentro del mismo palo. Otro método es **esAsignada()**, usado para indicar si a la carta se le han asignado valores, ya que indica que no ha sido asignada si el atributo valor del objeto es el mismo que en su declaración (0).

Jugador

Es una clase dedicada a las acciones y atributos de todos los jugadores, como sus 13 cartas de la mano y posibles movimientos. Se estructuran instanciándose en la clase MesaJuego y esa clase los gestiona y responde ante sus datos. Es por esto que los métodos y atributos que tiene son de objeto como su **constructor**, o métodos de obtención de atributos como **getCartas()**, **getCarta()**, en la que obtiene una carta dada su índice, **getCantidadCartas()** o **getImagen()** y métodos de asignación de estos, como **setCartas()** y **setImagen()**.

Otros métdos que tiene esta clase son los de realizar jugadas, como **jugadaBot()** que realiza una jugada en forma de jugador máquina, buscando una carta que contenga y pueda ser usada según las cartas de la mesa a través del método de validación **cartaValida()** de esta clase y que indica se se podrá o no usar una carta. También está el método **jugadaManual()** que representa una jugada por parte de un jugador humano, en el que no busca una carta a

usar sino que ya le es dada desde la clase MesaJuego y también es validada con el método de validación anterior.

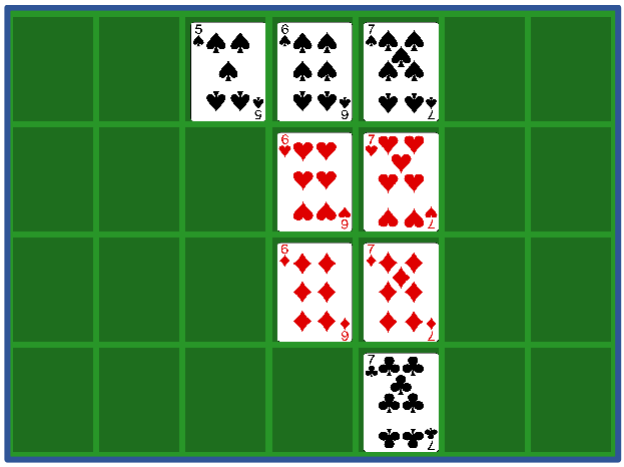
MesaJuego

Esta clase está destinada a la gestión y control de acciones a nivel de tablero o mesa de juego durante toda la partida, siendo esta clase instanciada en la clase principal y usada desde allí. Par ello cuenta con con métodos como el **constructor** para ser instanciada, o métodos para obtener sus atributos como **getTurnoJugador()**, **getJugador()** y **getCartas()**, que sirven para obtener los datos con los que actualizar la ventana gráfica.

Los métodos más funcionales son **barajarCartas()**, que obtiene el conjunto de las 52 cartas barajadas mediante la clase Baraja. Tiene método **repartirCartas()** que reparte las cartas que tenga entre las manos de los diferentes jugadores. Para que se juegue un turno está **jugarTurno()**, que se encarga de cambiar las cartas del jugador al que le toque y de la mesa, si se ha podido seleccionar una carta válida, y devuelve esta para actualizar la ventana de juego. Así también cuenta con el método **pasarTurno()** que se usa para avanzar el atributo entero que representa el turno del jugador activo. El encargado de gestionar el final de la partida es **haGanado()**, que comprueba si el jugador actual no tiene cartas y por tanto ha ganadolectura del fichero del diccionario que al estar formado por palabras, se trabaja a este nivel, de forma que cuenta con su respectivo constructor para poder instanciarlo como objeto con la ruta del fichero.

PanelCarta

Esta clase hereda de JPanel y es muy similar a BotonCarta, pero esta encargada de representar a las cartas no interactuables en el apartado gráfico del tablero. Los casos para instanciar este objeto son en la visualización de las cartas de la mesa, no interactuables y los indicadores de cartas de los jugadores máquina.



Para su uso, tiene los métodos **constructor**, para instanciarse, y el método sobreescrito **paintComponent()**, que gestiona el apartado gráfico al pintar la imagen de la carta sobre un rectángulo más oscuro como fondo haciendo uso de la clase Graphics2D.

Los métodos más funcionales son 2 **setImagen()**, para obtener la imagen de la carta dada una imagen como tal o el nombre del fichero de imagen. También está **reiniciar()** que

aplica valores por defecto al objeto, y otros métodos para aplicar valores como **setVisualizar()**, para indicar si se debe visualizar la imagen o no, y **setValor()** para indicar el valor numérico que debe mostrarse. Este último valor también se gestiona con un atribtuo constante que indica si se debe mostrar o no el texto con dicho valor.

TextoJugador

Esta clase que hereda de JPanel sirve para manejar y visualizar el contador del jugador humano, de forma que es una clase pequeña y sencilla. En ella está en métdodo sobreescrito **paintComponent()** que, através de Graphics2D, pinta con unos ajustes definidos el texto con el valor del indicador. Cuenta también con los métodos **reiniciar()** para ajustar sus datos a valores iniciales como sería poner el valor en 0 y el método **setValor()** que le aplica el valor numérico dado.

Conclusión

Al realizar la práctica, he podido ver la importancia y complejidad principalmente de la composición del apartado gráfico y visual, con el apartado lógico que se encarga de llevar los datos. También he podido ver las diferentes formas que llevar a cabo el mismo proceso, ya que en el caso de la visualización gráfica he optado por la opción de usar la clase Graphics o más concretamente Graphics2D, pero hay otras soluciones como aplicar iconos a los propios elementos de la interfaz. También destaco la forma de implementar el diseño descendente y la orientación a objetos al conectar los datos de, por ejemplo, las cartas de los jugadores que sirven para todos ellos, con sus representaciones gráficas haciendo uso de las interfaces.

Aún viendo complicaciones en apartados como la disposición de la interfaz gráfica algunas veces, o principalmente al hacer las interacciones del jugador con los elementos de la ventana a través de los gestores de eventos, resolviendo los diferentes problemas de forma estructurada no ha habido más problemas. Al hacer uso de una estructura previa con la que hacer paso por paso las partes del programa, se modulariza más y los problemas que surgen no se hacen complejos, como sí podrían hacerse en el caso de tener muchas secciones y conectadas si orden.

De esta forma que he visto sobre todo complicaciones a la hora de plantear el diseño de la estructura y organización del programa, ya que al haber varias formas de obtener el mismo resultado podía requerir de unas necesidades u otras, como clases que necesitasen atributos de clase o de objeto, como ha resultado en la clase Baraja. Esto es porque al originalmente la había planteado como clase a instanciar, pero al ver que no era lo más deseable acabó siendo una clase con atributos y métodos estáticos.

Al ir trabajando sobre las diferentes partes, sobre todo al final, ya las podía unir hasta finalizar y, después de una par de pruebas y ayuda de los comentarios, conformar el programa con el resultado esperado.