UNIVERSIDAD DE MARGARITA

ALMA MATER DEL CARIBE

VICERRECTORADO ACADÉMICO

DECANATO DE INGENIERIA

UNIDAD CURRICULAR: ESTRUCTURA DE DATOS

INFORME PROYECTO 1: JUEGO DE MEMORIA

Profesor:

Cesar Requena

Realizado por:

José Ferreira C.I: V-28.315.655

Paola Márquez C.I: V-27.125.784

El Valle del Espíritu Santo, 23 de octubre del 2023

Para las etapas iniciales del diseño y elaboración del juego de memoria, fue necesario primero pensar cuales son los elementos que lo conforman. El primero y el más obvio, son las cartas, que se decidió que serían 18 en total para tener 9 pares iguales (número impar que el empate sea menos probable). Iniciamos creando una clase llamada “Carta” que contendrá los atributos de ID donde será almacenado un numero entero que lo distinga del resto de cartas y otro atributo entero estático llamado “paresEncontrados” para llevar registro de los pares de cartas que han sido encontrados durante el juego, además de su constructor que recibe el parámetro para guardar el valor de la ID en la carta creada, junto a los métodos de la clase para retornar el ID de la carta, los pares encontrados y sumar 1 par encontrado. Otro elemento necesario en un juego son los jugadores, que se decidió que habría entre 2 y 4, por lo que se creó una clase “Jugador” con los atributos de nombre (cadena), puntaje (entero), turno (entero), ID (entero) y un atributo estático entero que sería la cantidad de jugadores, su constructor para darle valores iniciales a los primeros 4 atributos mencionados, además de los métodos para retornar cada uno, establecer la cantidad de jugadores y sumar un turno y un punto al jugador.

El siguiente paso fue pensar cómo se manejarían estos datos, específicamente, cuál estructura se usará para almacenarlos y manipularlos. Para las cartas, se sintió lógico e intuitivo utilizar pilas, debido a que en la vida real cuando las usamos solemos ponerlas en pilas. Es entonces como se dio inicio al desarrollo de la clase “Nodo” con el atributo que almacenará la información de cada carta como objeto y el nodo que le sigue, junto con los métodos setter y getter para establecer y retornar la carta que será almacenada tanto en el nodo actual como en el siguiente. Estos nodos serán usados para la implementación de las pilas, planteadas en la clase “Pila” con su atributo de nodo y tamaño, con los métodos para determinar si está vacía, retornar su tamaño, agregar una carta nueva al tope de la pila y sacarla de la pila. Más adelante será necesario guardar ID con valores enteros dentro de una pila, por lo que creamos los atributos y métodos necesarios para almacenar y manipular valores enteros con los nodos en una pila.

Para crear las cartas de manera eficiente se usó un arreglo de 18 elementos y usando un for loop va creando los objetos de la clase carta con el mismo ID para la carta en la posición i e i+1 hasta llenar el arreglo. Cabe destacar que esto crea las cartas con los ID organizados de menor a mayor y necesitamos que las cartas estén “barajeadas” en un orden aleatorio. Para ello, se creó una clase “cardRandomizer” con un único método estático “randomize” que recibe como parámetro el

arreglo de cartas ordenadas y usando un generador de números de forma aleatoria y una arraylist que posee los índices del arreglo, va ingresando las cartas del arreglo en una pila generada. De esta forma, conseguimos una pila que tiene las cartas almacenadas en un orden aleatorio.

Para almacenar y manipular los objetos de la clase jugador, se decidió usar la cola, debido a que son fáciles de usar y permite llevar la secuencia de turnos de una manera intuitiva, en la que cada jugador debe llegar al inicio de la cola para que sea su turno y una vez lo termina, se devuelve al final. Es así como se crea la clase "Cola", con los atributos de los nodos que van en el inicio, en el final de la cola y su tamaño, con su constructor que inicia cada cola con valores nulos y los métodos para determinar si esta vacía, agregar un jugador al final de la cola, sacar al jugador al frente de la cola y un método llamado "ordenarPorPuntaje" que toma la cola de jugadores y devuelve una cola de jugadores ordenada de mayor a menor puntaje usando un método parecido al de selección, de manera que el jugador con el puntaje más alto está al inicio de la cola.

Para la cola de jugadores se usan los mismos nodos de la clase que ya está creada, pero para eso, primero fue necesario agregar atributos y métodos específicos para los valores de la clase Jugador.

El siguiente paso fue crear el JFrame Form con el nombre "Ventana" donde se mostrará la interfaz gráfica y los distintos elementos que podrá ver y con los que podrá interactuar el usuario. Para el diseño del interfaz se usarán las herramientas de diseño del JFrame Form con las que cuenta el NetBeans. Para la ejecución del programa se usará el main en el código generado de la interfaz y todo el código con la parte lógica estará dentro del constructor de la ventana para que se ejecute al iniciar ese main que posee una llamada al constructor.

El programa inicia creando las 18 cartas en un arreglo y luego almacenándolas de manera aleatoria en una pila llamada ranCards. Luego solicita al usuario la cantidad de jugadores que jugaran, pudiendo escoger entre 2, 3 y 4. La opción seleccionada será el valor ingresado como parámetro de setCantidad() de la clase Jugador. Después crea los objetos de la clase jugador para ser introducidos dentro de la cola jugadores con el nombre ingresado y asignándole una ID de manera automática. Una vez terminado esos procesos, se inician los componentes visuales de la ventana.

Para plasmar los distintos componentes visuales en la ventana se usarán jLabels. Lo primero que se hizo fue el diseño y la distribución de las cartas en la ventana. Se consiguió un set de imágenes de cartas en internet y se ajustaron al tamaño deseado para ser usados como iconos para las cartas. Cada imagen es agregada a un paquete llamado “Imágenes” y guardada dentro de una variable de tipo ImageIcon para ponérselas a los jLabels usando el método setIcon() con la dirección de la imagen. Todas las cartas deben empezar volteadas, por lo que la imagen inicial de todas es la que corresponde al reverso de la carta.

Para el fondo se usó una imagen en un jLabel con las dimensiones de la ventana y que esté por debajo de todos los componentes y para la información sobre el nombre, imagen, turno y puntos de cada jugador se usan jLabels que se muestran dependiendo de la cantidad de jugadores y actualizando por medio de un procedimiento que será explicado más adelante.

Cada acción del usuario se dará mediante un click, por lo que se vinculó un evento de “MouseClicked” a cada jLabel que contiene una carta. Cada uno de estos eventos tendrá 2 procedimientos: comparar() y finalizar(). El procedimiento comparar se ejecuta sólo cuando una variable booleana llamada “enableClick” es true y cuando la imagen en el jLabel es el reverso de la carta. El mismo recibe como parámetro el jLabel al que se le hizo click y como primer paso muestra la imagen que corresponde al ID de su carta introduciéndolo en un switch. Para vincular cada jLabel de carta con un objeto de la clase Carta, se aprovechó la propiedad que tienen los jLabel de almacenar texto, por lo que al iniciar cada jLabel se le asigna en el campo de texto la ID convertida en cadena de la carta que fue sacada del tope de la pila de cartas aleatorias (de esta forma, las IDs también son asignadas aleatoriamente). El procedimiento revisa la ID que se encuentra en el texto del jLabel y dependiendo de su valor, el switch le asigna su icono correspondiente. Luego este jLabel es almacenado en una variable para ser manipulado en el próximo click y su ID es introducido en una nueva pila llamada “pareja” que será usada para las comparaciones.

Cuando el jugador le de click a la segunda carta, se repetirá el procedimiento anterior con el nuevo jLabel y el tamaño de la pila “pareja” será 2, en ese momento se procede a comparar ambos ID dentro de la pila. Si hay match, las cartas se quedan volteadas, se le suma un punto al jugador, se agrega 1 al contador de pares encontrados y si es su primer punto, se agrega a una cola llamada “primerPunto” cuya finalidad será explicada más adelante. En caso de no haber match, las cartas

se quedan volteadas por 1.5 segundos y se desactivan los clicks (enableClick=false) para que el usuario no pueda interactuar con las cartas en ese momento. Luego de que se cumple el tiempo, la carta clickeada y la carta anterior que fue almacenada en la variable se voltean, se reactivan los clicks, se le suma un turno al jugador, se actualiza su información en los jLabels donde aparece sus puntos y turnos usando el procedimiento actualizar() y es agregado nuevamente al final de la lista de jugadores. El proceso actualizar toma el ID del jugador que está en su turno y dependiendo de ese valor, actualiza los datos de los jLabel en el recuadro del jugador correspondiente usando un switch.

El proceso finalizar solo se ejecutará una vez los nueve pares de cartas hayan sido encontrados (Carta.getParesEncontrados()==9). El primer paso es usar el método ordenarPorPuntaje para ordenar la cola de jugadores por puntaje y almacenarla en la cola “ranking”, cada uno de estos jugadores es guardado en variables en el orden en que salen de ranking, siendo el jugador con el mayor puntaje el primero. El proceso revisa la cantidad de jugadores y dependiendo de cada caso chequea si hay empate en el primer lugar o no, comparando puntajes. En caso de no haber empate, muestra nombre, puntaje y turnos del jugador que está de primero en el ranking. Si hay empate, usa la cola “primerPunto” para ver cuál fue el primer jugador entre los que están empatados que consiguió su primer punto primero. El que consiguió antes su primer punto, gana el desempate.

En cada uno de estos casos, además de anunciar el ganador, también genera un archivo de texto en la carpeta del proyecto llamado “Ganador.txt” con la información anteriormente mencionada del ganador del juego. Para esto se creó una clase llamada “ArchivoTexto” con un único método estático llamado “crear()” que genera el archivo.

Finalmente, cada acción o evento está acompañado de un efecto de sonido o música. Para conseguirlo se hizo una clase llamada “Reproductor” que posee los métodos estáticos que reproducen cada sonido y son llamados cuando se realiza la acción correspondiente.