Implementacion del juego "El solitario"

Primer proyecto de Estructuras de datos

1 de mayo de 2017

Integrantes:

- *Ferro Palomino, Gianfranco Augusto*
- *Ipanaqué Casquina, Ingrid Fransheska*
- *Hidalgo Chávez, Daniel Alfredo*
- *Huacac Chiri, Claudia Esteffani*
- *Aguirre Janampa, Cristian Fernando*

INTRODUCCIÓN

La programación estructurada desempeña un papel de elevada importancia en la informática, al igual que en el aprendizaje de todo aquel que estudia programación, y como toda disciplina presenta un sinfín de problemas que por si mismos suponen un desafío en el avance de esta. Para un óptimo entendimiento de esta disciplina se requiere de sólidos conocimientos en lógica, capacidad de abstracción y de la buena práctica de ordenamiento de datos.

El proyecto que mostraremos a continuación está basado en el lenguaje de programación C, ya que éste lenguaje es multiplataforma y funciona en diferentes sistemas operativos, donde pondremos en práctica conceptos como estructuras, arreglos, pilas y muchos conceptos más aprendidos a lo largo de nuestras clases necesarios para la implementación de nuestro proyecto.

OBJETIVOS

- Programar el juego "Solitario".

- Desarrollo y aplicación de conocimientos de estructuración de datos, arreglos y pilas en lenguaje C.

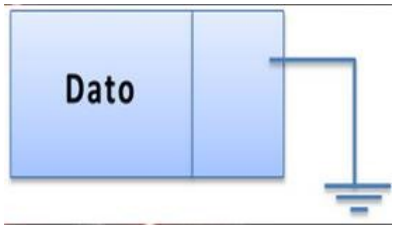
- Desarrollo de una interfaz gráfica.

- Obtener nuevos conocimientos de la estructuración de datos a través de la investigación y el desarrollo del proyecto.

MARCO TEORICO

- Nodos

Una lista se compone de nodos, que son estructuras de datos que nos permiten registrar datos de interés. Para que estos nodos se conviertan en una lista, debe existir un enlace entre ellos, que en términos mas propios se los conoce como apuntadores o punteros. Los punteros o apuntadores, son la parte de los nodos que nos permiten recorrer la lista y acceder también a las direcciones de memoria donde se almacenan los elementos de la lista.



- Pila

Una pila (stack) es un tipo fundamental de lista (Tipo de dato abstracto fundamental) en la que todas las inserciones y supresiones de elementos solo se pueden realizar desde una única posición, el extremo, la cima o tope de la pila (TOP)

Las tres operaciones más importantes un una pila son:

PUSH: Meter, poner, apilar. Operación que inserta un elemento en la cima de la pila.

POP: Sacar, quitar, desapilar. Operación que elimina el elemento que ocupaba la posición cima. El elemento que había sido insertado inmediatamente antes que el eliminado pasara a ser ahora la nueva posición cima.

TOPE: Operación que devuelve el elemento que ocupa en ese instante la posición cima pero sin extraerlo.

- Estructuras

Las estructuras conocidas generalmente con el nombre de registros, representan un tipo de datos estructurado. Se utilizan para resolver problemas que involucran tipos de datos estructurados, heterogéneos.

Ünа estructura es una colección de elementos finita y heterogénea"

Finita, porque se puede determinar el número de componentes y heterogénea porque todos los elementos pueden ser de tipos de datos diferentes. Cada componente de la estructura se denomina campo y se identifica con un nombre único.

Ejemplo:

```
typedef struct Alumno {
        char num_carne[11];
        char nombre_alumno [60];
        int edad;
        char direccion[60];
} Alum;
```

- Algoritmo de colisiones

Podemos definir una colisión entre dos elementos como un choque entre estos dos elementos. En el mundo del videojuego el significado de esta palabra es el mismo añadiéndole algunos matices. Se produce una colisión cuando dos personajes u objetos del videojuego chocan entre ellas.

La detección de colisiones es una técnica que consiste en implementar unas funciones que nos permitan conocer cuando ha existido una colisión entre dos elementos. Este va a ser el núcleo de nuestro trabajo en cuanto a las colisiones.

Existen varias formas de solventar en cierta medida los problemas que acarrea el que las superficies sean rectangulares. La primera de ellas es recortar todo lo posible el lienzo sobrante así tendremos menos exceso por lo que el comportamiento de la colisión será más adecuado. Esto muchas veces no está en nuestras manos. Como programadores, y en proyectos de cierta envergadura, no nos haremos cargo del diseño de los personajes y seguramente no tengamos tiempo de ajustar todos los lienzos que pueden existir en una animación con el trabajo que supone retocar una rejilla de imágenes.

La segunda forma de evitar un mal comportamiento de las colisiones es realizar una buena implementación de las funciones dedicadas a detectar dichas colisiones. Podemos añadir información a nuestro personaje que ciña la superficie de colisión lo más posible al personaje para conseguir una mejor respuesta. Esta es la opción más pausible y la que vamos a desarrollar.

En definitiva, una colisión es un solapamiento de píxeles al que, normalmente, deberemos de reaccionar. Para que esta reacción se haga en el momento correcto tendremos que implementar unas funciones dedicadas a la detección de colisiones que veremos a continuación.

- Algoritmo Knuth-Morris-Pratt

El algoritmo KMP, trata de localizar la posición de comienzo de una cadena, dentro de otra. Antes que nada con la cadena a localizar se precalcula una tabla de saltos (conocida como tabla de fallos) que después al examinar entre si las cadenas se utiliza para hacer saltos cuando se localiza un fallo.

Supongamos una tabla 'F' ya precalculada, y supongamos que la cadena a buscar esté contenida en el array 'P()', y la cadena donde buscamos esté contenida en un array 'T()'. Entonces ambas cadenas comienzan a compararse usando un puntero de avance para la cadena a buscar, si ocurre un fallo en vez de volver a la posición siguiente a la primera coincidencia, se salta hacia donde sobre la tabla, indica el puntero actual de avance de la tabla. El array 'T' utiliza un puntero de avance absoluto que considera donde se compara el primer carácter de ambas cadenas, y utiliza como un puntero relativo (sumado al absoluto) el que utiliza para su recorrido el array 'P'.

Se dan 2 situaciones:

Mientras existan coincidencias el puntero de avance de 'P', se va incrementando y si alcanza el final se devuelve la posición actual del puntero del array 'T'

Si se da un fallo, el puntero de avance de 'T' se actualiza hasta, con la suma actual del puntero de 'P' + el valor de la tabla 'F' apuntado por el mismo que 'P'. A continuación se actualiza el puntero de 'P', bajo una de 2 cicunstancias; Si el valor de 'F' es mayor que -1 el puntero de 'P', toma el valor que indica la tabla de salto 'F', en caso contrario vuelve a recomenzar su valor en 0.

Pseudocódigo del algoritmo de KNUTH

```
Algoritmo BúsquedaKMP:
  Entrada:
    un array de caracteres, T (el texto donde se busca)
    un array de caracteres, P (la palabra/s que se busca)

  Salida:
     un entero que expresa la posición en T en la cual se encontró P.
        (nota: opcionalmente puede convenir devolver un entero con signo).

  Definición de variables:
     un entero, k <- 0 (puntero de examen en T)
     un entero, i <- 0 (la posición del carácter actual en P, y avance
   relativo respecto de k, para T)
     un array de enteros, F (la tabla de fallo, calculada a continuación, o
   en otra parte)

   Si tamaño de T es mayor o igual que tamaño de P entonces
      Precalcular TablaKMP(P,F)
      Mientras k + i es menor que la longitud de T, hacer
          Si P[i] = T[k + i] entonces
             Si i es igual a la longitud de P - 1 entonces
                Devolver k
             Fin si
             Asignar i <- i + 1
          Si no entonces
             Asignar k <- k + i - F[i]
             Si i es mayor que 0 entonces
                Asignar i <- F[i]
             Fin si
          Fin si
      Repetir
   Fin si

   (si se alcanza este punto, se buscó en todas las T sin éxito)
   Devolver longitud de T, ó si se usa una variable con signo, devolver  -1
```

DESARROLLO

El desarrollo del proyecto se definirá de la siguiente manera:

La aplicación que se desarrollará en el lenguaje de programación C, es la simulación de un juego llamado .[El] solitario", lo cual es una aplicación en el dominio de pilas como una estructura de datos.

– **Distribución de las cartas**

El total de pilas que desarrollaremos serán 13 en total, de las cuales están distribuidas de la siguiente manera:

. 7 pilas de juego
. 4 pilas de salida
. 1 pila de reserva
. 1 pila de descarte

El juego consta de una baraja de 52 cartas de las cuales distribuiremos 28 de las cartas en **las 7 pilas de juego** de la siguiente manera:

. En la primera pila 1 carta
. En la segunda pila 2 cartas
. Y así sucesivamente hasta tener 7 cartas en la séptima pila.

La condición es que en las 7 pilas, siempre la última carta este visible o abierta, como se ve en la figura de abajo:

Las cartas restantes son colocadas **cerradas** en otra pila llamada **pila de reserva**.

Por otro lado tenemos, la **pila de descarte**, la cual no tendrá ninguna carta al comienzo del juego y estará a la espera de cartas **abiertas** de la **pila de reserva**.

Por ultimo tenemos 4 **pilas de salida**, la cual estará inicialmente vacía y a la espera de poder llenarse con las condiciones que detallaremos más adelante.

IMPLEMENTACIÓN DE LAS PILAS:

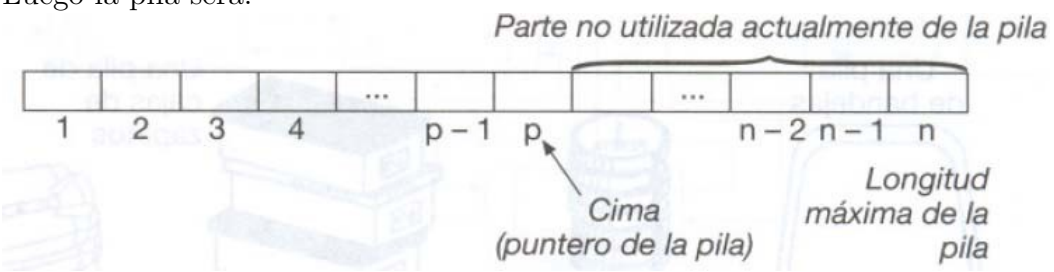
. Mediante vectores:

Una pila puede implementarse mediante vectores, en cuyo caso la dimensión es fija. En este caso se dirá que es una implementación estática. Cuando lo implementamos mediante un vector hay que tener en cuenta que el tamaño de la pila no puede exceder el número de elementos del array.

Definiremos, antes de implementar una pila mediante vectores, el número máximo de elementos que podremos meter en el apila. Será n La declaración de pila es:

```
Elemento = E;
Pila = registro de
       cima: numérico;
       arreglo: vector[1...n] de elemento:
       fin registro;
```

Luego la pila será:



Donde cima corresponde a la posición en el vector del último elemento insertado en la lista, o sea que a de apuntar en todo momento al elemento tope. Así si pila.cima=01 entenderemos que la pila esta vacía, mientras que si pila.cima = n la pila estará llena. Esta implementación no es apropiada cuando no se conoce el número máximo de elementos que puede tener una pila ya que

podría quedarse corta por faltarle posiciones, y se produciría un error, o por pasarse de larga, es decir, utilizar mucha más memoria de la estrictamente necesaria y desaprovecharla. De todas formas, esta implementación tiene una gran ventaja, que es la sencillez y rapidez con la que se programa y se realizan las operaciones.

Por ultimo tenemos 4 pilas de salida, la cual estará inicialmente vacía y a la espera de poder llenarse con las condiciones que detallaremos más adelante.

– **Movimientos admitidos de las cartas**

. En las pilas de juego solo se pueden abrir cartas que formen una secuencia decreciente consecutiva y de colores alternados, por ejemplo si en el tope de la pila se encuentra el 7 de trébol, solo se puede colocar sobre él un 6 de oro o de corazones.

. Todas las cartas abiertas de la pila de juego pueden ser movidas como una unidad y colocadas en otra pila desde que cumplen la regla anterior.

. Cada pila de juego debe tener por lo menos una carta abierta, de modo que si una carta abierta fue removida, entonces la carta del tope debe ser abierta.

. Cuando una pila de juego queda vacia ella solo puede recibir o un rey que esta abierta en alguna pila de juego(conjuntamente con las cartas sobre él) o la carta que se encuentra en el tope de la pila de descarte.

. Siempre que un As aparece en el tope de una pila de juego o de descarte ella debe ser pasada directamente a una de las pilas de salida.

. En cada una de las pilas de salida las cartas son colocadas abiertas, formando una secuencia consecutiva creciente del mismo naipe. No está permitido remover las cartas de la pila de salida. Solo una carta del tope de la pila de descarte o de la pila de juego debe ser colocada en la pila de salida.

. Las cartas de la pila de reserva pueden ser removidas abiertas hacia la pila de descarte uno a la vez.

. Las cartas de la pila de descarte solo pueden ser removidas una a la vez. Siempre que la pila de descarte quede vacía una carta de la pila de reserva se transfiere a ella.

. Para empezar el juego, mover una carta de la pila de reserva a la pila de descarte.

. Si no es posible realizar ningún movimiento transferir una carta de la pila de reserva a la pila de descarte.

. El juego termina cuando no es posible realizar ningún movimiento y la pila de reserva quede vacía.

– **Caracteristicas del programa**

. El juego puede efectuarse varias veces que el usuario desee.

. Las 13 pilas usadas para el programa están almacenadas en un vector de 60 posiciones, con las operaciones de apilar, desapilar, pilavacia, etc.

. Las cartas están divididas en 4 tipos: corazones, espadas, diamantes y tréboles, para tipo de carta hay una secuencia de números: A(as), 2, 3, 4, 5, 6, 7, 8, 9, 10, 111, 12, 13. Asi como su color: negro o rojo y también si carta esta visible o no.

REFERENCIAS

- The art of computer programming, Knuth Donald, editorial: Addison-Wesley vol. 1.