Juego **MEMORIA**

Lenguajes de Programación - 21721

Práctica final en LISP

Profesor:

Juan Montes de Oca

Autores:

A. Antonio Boutaour Sanchez - 43202227Q

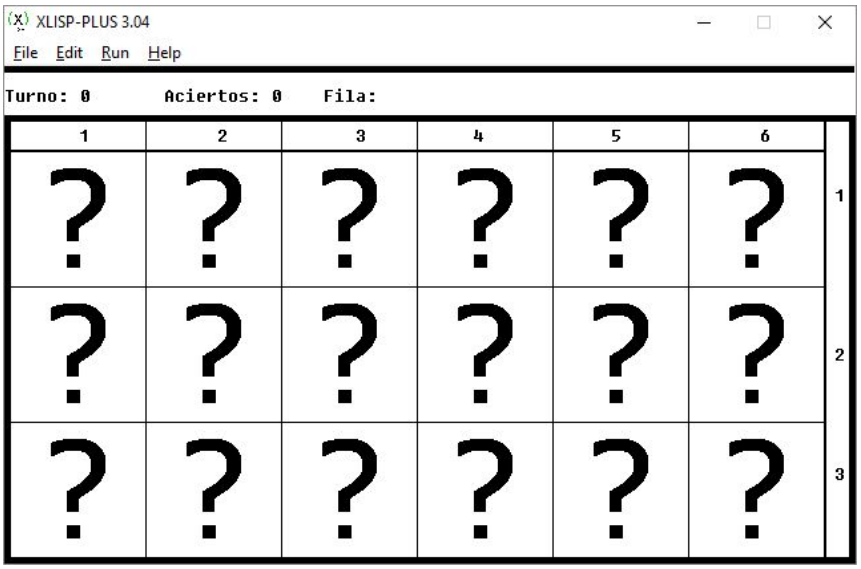
Bernat Pericàs Serra - 43212796M

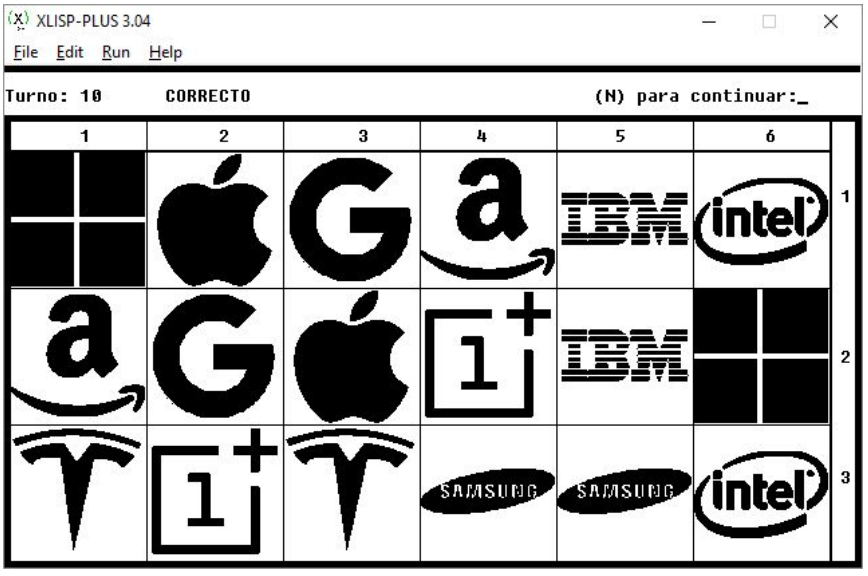
Universitat de les Illes Balears

Índice

Introducción

El problema a resolver es la implementación del conocido juego de memoria basado en parejas de imágenes o figuras. Esta simulación debe realizarse en LISP y contará con 18 imágenes (9 parejas) representadas aleatoriamente en una matriz 3 x 6. En todo momento debe llevarse el recuento del turno y, al finalizar la partida deberá quedar registrado junto al nombre del jugador.





Generación de imágenes a partir de BMP

Para realizar la generación de las imágenes BMP se ha usado la gran capacidad de tratamiento de datos del lenguaje Python.

El programa realiza para todos los elementos de la carpeta /bmp una selección de uno de los tres canales RGB, y se encarga de realizar un fichero .IMG con la representación del Byte leido. Estos ficheros son almacenados en la carpeta /img.

Se representa de la siguiente manera:

```
import cv2
#import numpy as np
import struct
import os
from os import listdir
from os.path import isfile, join

bmp_path =  'C:/Users/Toni/Documents/02-UIB/17-18/2-Quatri II/Llenguatges de programacio/Practica/Imagenes/bmp/'
img_path= "C:/Users/Toni/Documents/02-UIB/17-18/2-Quatri II/Llenguatges de programacio/Practica/Imagenes/img/"

data =[]

data_bmp= [f for f in listdir(bmp_path) if isfile(join(bmp_path,f))]

for a in data_bmp:
    bmp_name = os.path.splitext(a)[0]
    print "la imagen a tratar es " + bmp_name
    f_bmp = join(bmp_path,(os.path.basename(a)))
    x = cv2.imread(f_bmp)
    _, _, r = cv2.split(x)
    f_img = join(img_path, bmp_name+'.img')
    with open(f_img, 'wb') as f:
        for c in r:
            for b in c:
                f.write(struct.pack("B",b))
```

Tutorial

El uso del programa es sencillo, una vez cargadas las funciones del fichero *PRACTICALISP.lsp*, bastará ejecutar la función *inicio()* introducir un nombre de usuario y aparecerá por pantalla el tablero con las 18 imágenes ocultas listas para ser descubiertas.

El contador de turnos y aciertos se sitúan arriba, junto al espacio para introducir por teclado el número de fila y columna de la imagen a descubrir (tras introducir el número de fila/columna se debe presionar el botón *Enter*). Una vez elegidas dos imágenes, se indicará por pantalla si la elección es correcta (pareja válida) o no y se dará la oportunidad de seguir jugando introduciendo la letra N y *Enter*. Al haber descubierto todas las imágenes aparecerá un mensaje por pantalla confirmándolo. Después el usuario podrá decidir empezar otra partida o no.

Código

Estructura principal (función *inicio ( )*)

La estructura del programa se puede observar en la función *inicio(),* con la que se inicia el juego y que continúa ejecutándose hasta el final de la ejecución.

El primer paso es abrir el flujo de salida al fichero donde se guardaran los resultados, preguntar al usuario su nombre, pintar el panel del juego, generar aleatoriamente un array de imágenes y llenar el panel de imágenes de tipo reverso (al principio las imágenes están ocultas).

Después de estos pasos iniciales, se entra en el bucle principal con la instrucción *loop.* Este siempre empezará pintando (o actualizando) el número de turno, el contador de aciertos y las elecciones del jugador (número de fila, número de columna, imagen correspondiente).

A continuación, con la función *comprobar_casillas()* se decidirá si el movimiento elegido por el usuario descubre una pareja válida o no y se ejecutará una parte del código u otra; actualizar el array de imágenes para reflejar la pareja descubierta (sólo en caso de pareja válida), mostrar al usuario si la pareja es correcta o incorrecta, escribir en RESULTADOS.txt el movimiento, esperar que el usuario confirme el siguiente movimiento, incrementar el contador de parejas reveladas (en caso de pareja válida) y girar las imágenes reveladas (solo en caso de pareja no válida).

Finalmente, el bucle acabará incrementando el número de turnos y comprobando si el *contador* es igual a 9, es decir, si todas las 9 parejas han sido descubiertas.

Funciones

movimiento ( )

Esta función se encarga de gestionar todo lo relacionado con la elección de dos casillas por parte del jugador. Primero lee la fila y columna que el usuario introduce, comprueba que se corresponda a una casilla del tablero y que no haya sido revelada. Si todo es correcto gira la casilla revelando su imagen y se repite el proceso otra vez, pero comprobando, además, que no se elija la misma casilla que la anterior ya revelada.

espera ( )

Se encarga de parar el bucle principal al final de cada turno, cuando se han seleccionado dos imágenes (sean una pareja válida o no) hasta que el usuario desee continuar (introduciendo la tecla N).

pintar_turno ( ) y pintar_contador ( )

Estas dos simples funciones sirven para mostrar por pantalla el turno actual y el número de aciertos hasta el momento.

poner_a_false(fil1 col1 fil2 col2)

Refleja en en array la revelación de una pareja de imágenes poniendo a 0 su casilla.

tratamiento_errores (a)

Imprime el error correspondiente al parámetro de entrada *a.* Los tres errores propios que se han definido son: “FORMATO INCORRECTO”, “MISMA CASILLA” y “CASILLA YA REVELADA”.

comprobar (a b)

Comprueba que el número de fila y columna tenga un formato válido (sea una casilla del tablero 3x6) y que no esté ya revelada. La primera condición son simples operaciones matemáticas y la segunda operaciones lógicas usando la función *aref*.

comprobar_casillas (a1 b1 a2 b2)

Comprueba que la pareja de casillas seleccionadas sea una pareja de imágenes válida accediendo y comparando el array de imágenes *arr_img*.

son_iguales (a1 b1 a2 b2)

Comprueba que las dos casillas introducidas sean la misma o no comparando sus componentes fila (a1 y a2) y columna(b1) y b2.

generar_arr_img ( )

Genera un array de imágenes aleatorio haciendo uso de la función *random* y *apariciones().*

apariciones (b a)

Esta función se usa en *generar_arr_img ( )*. Comprueba que sólo aparezcan dos casillas de cada imagen para no duplicarlas en el array aleatorizado de imágenes.

poner_un_reverso (fil col)

Pinta un reverso en la casilla pasada por parámetro. La imagen reverso es una imagen especial representada por un interrogante, por lo que se usa la función *leer_imga (posx posy)* igual que para las imágenes de las parejas.

poner_reversos (fil1 col1 fil2 col2)

Hace uso de la función *poner_un_reverso (fil col)* para pintar dos reversos, útil al finalizar un turno que no haya revelado una pareja válida.

todo_reversos ( )

Pinta todo el panel con reversos, útil al inicio del juego cuando se dibuja el tablero con ninguna pareja descubierta.

girar_carta (fil col)

Se encarga de revelar una única casilla pintando la imagen que le corresponde haciendo uso de la función *leer_img (a posx posy)*.

felicitacion ( )

Se encarga del mensaje del final de la partida donde se felicita al usuario y se le pregunta si desea terminar el juego (tecla N) o iniciar otra partida (tecla S).

select_nombre ( )

Permite al usuario introducir su nombre para que pueda quedar registrada su puntuación en el fichero RESULTADOS.txt.

pintapanel ( )

Pinta el panel haciendo uso de las funciones *pintalinea_vert (x y1 y2 n), pintalinea_hor (x1 x2 y n), pinta_g_h ( ), pinta_g_v ( ).* Es una tarea laboriosa y manual.

pinta_g_h ( ) y pinta_g_v ( )

Pinta las líneas de la tabla horizontales y verticales.

pintalinea_hor (x1 x2 y n) y pintalinea_vert (x y1 y2 n)

Pintan una línea horizontal o vertical desde el punto pasado por parámetro y un tamaño también especificado.

limpiar_img (posx posy)

Es la función que se encarga de limpiar el área correspondiente a una imagen o casilla pintando de blanco esa región (100x100 píxeles) usando dos bucles *dotimes*.

pintar (x y)

Simplemente pinta un píxel en la coordenada pasada por parámetro, primero debe moverse a esa posición.

leer_img (a posx posy)

Lee byte por byte el fichero .bmp y dibuja la imagen pintando el píxel solo si el byte leído (componente roja) del fichero se corresponde con 255 o no.

Conclusión

La realización de esta práctica nos ha hecho ver que no es necesario disponer de un lenguaje potente en cuanto a funciones gráficas para simular un juego. Basta con hacer un uso inteligente de las funciones gráficas elementales. Usando 5 funciones gráficas se puede crear un juego agradable a la vista y divertido.

Nos ha permitido aprender un lenguaje de programación muy diferente de los que estamos acostumbrados a usar, sin entorno propio ni control de errores hace muy difícil la búsqueda de errores y depuración del código, aún así, con paciencia, se puede conseguir lo propuesto.

Como problema principal surgido de la programación en Lisp ha sido el hecho de que la primera tanda del juego no llega a ser aleatorio, puesto que cumple siempre la misma distribución inicial, pero la segunda ejecución actúa como se espera, generando un nuevo juego de posiciones que realmente es aleatorio. No se ha sabido solucionar este problema y se piensa que se trata de un problema de la propia función *random*.

Cabe destacar que se ha realizado el tratamiento de todos los posibles errores de introducción de texto, siendo imposible gestionar que la tecla *enter* genere un salto de línea, afectando a la visualización del juego. Se trata de una limitación del propio lenguaje.

A pesar de estas complicaciones, hemos llevado a cabo la tarea propuesta y el resultado ha sido gratificante dadas las limitaciones de este lenguaje.