INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Cómputo

Periodo 2019-2

Proyecto Final: Videojuego "Pong" en Lenguaje C

(4/Junio/2019)

Grupo: 1CV1

Unidad de Aprendizaje: Algoritmia y Programación Estructurada

Integrantes:

*Díaz Matus Ricardo

*Martínez Olivares Vicente Jafet

Resumen:

El presente proyecto tratará de resolver un problema, el cual consiste en la realización de un videojuego en Lenguaje C, tal videojuego lleva por nombre “PONG”. Con base a todo lo visto y aprendido en la Unidad de Aprendizaje de Análisis de Algoritmos y Programación Estructurada durante el curso, se pondrán en práctica dichos conocimientos. El objetivo del videojuego es evitar que la pelota supere la barra, entre más rebotes dé, más puntos se conseguirán. En el ambiente del videojuego, en la pantalla se mostrara una barra vertical/horizontal y una pelota iniciando a rebotar en toda la pantalla. Aquí el usuario, tendrá el control de la barra, mediante el teclado, las teclas que propician el movimiento de la barra son (W-S) (A-D). Y todo esto se implementarán bajo el lenguaje C, y por medio de estructuras básicas realizarlos introduciendo funciones como son los tipos de datos (float, int o char), de igual se usarán los gráficos, ya que son fundamentales para la ejecución del videojuego “PONG” y también el uso de archivos, donde se generarán nuevas partidas del mismo videojuego, como la partida anterior que se guardó, los cuales conllevará a solucionar dicho problema antes mencionado.

Algoritmos:

Pseudo-código

i) **Para el programa-videojuego “Pong”**

Primero se declaran las bibliotecas a utilizar, aquí se incluye la gráfica y la creación de hilos, la cual es pthread.

Paso 1: Se declaran 1 función void (void Init) y 4 funciones void con apuntador que son, void *bolita(void *args), void *cancha(void *args), void *barras(void *args),void *marcador (void *args).
Paso 2: Se declaran funciones de tipo entero.
Paso 2.1: Se dan valores para las dimensiones de la cancha y el tamaño de las barras.
Paso 2.2: Se declaran el marcador de los jugadores 1 y 2.
Paso 3: Se declaran 4 hilos pthread_t_proceso.
Paso 3.1: Se declaran la creación de 4 hilos, con los 4 procesos y las 4 funciones void, con un carácter NULL. Más la unión de 4 hilos más, con los 4 procesos más el carácter NULL.
Paso 3.2: Se cierra el gráfico.
Paso 4: Se llama a la función void *marcador(void *args).
Paso 4.1: Mientras 1==1. Se muestra el texto (XMax/2),650, "Marcador:")
Paso 4.1.1: Si (marc_p1 == 0 && marc_p2 == 0)
Paso 4.1.1.2: Se muestra el texto outtextxy((XMin + 20),660, "0");
Paso 4.1.1.3: Se muestra el texto outtextxy((XMax - 50),660, "0");

Paso 4.2: Se escribe un retardo de 300.

Paso 5: Se llama a la función void *bolita(void *args)
Paso 5.1: Se declaran 3 valores de tipo enteros que son las dimensiones de la pelota.
Paso 5.1.1: Se declara el color del contorno de la pelota (WHITE).
Paso 5.2: Mientras 1==1.
Paso 5.2.1: Se declara el color del contorno de la pelota (WHITE).
Paso 5.2.1.2: Se declara el color de relleno y trama (BLACK).
Paso 5.2.1.3: Se declara la función circle, para dibujar la pelota con los valores enteros anteriores.
Paso 5.3: Se declara x+=Dx, y+=Dy.
Paso 5.3.1: Si ((x==XMin)||(x==XMax)). Dx*= -1.
Paso 5.3.2: Si ((y==YMin)||(y==YMax)). Dy*=-1;
Paso 5.3.3: Si (x==105) && (y>=coordy1) && (y<=coordx1). Dx*= 1.5
Paso 5.3.4: Si (x==790) && (y>=coordy2) && (y<=coordx2). Dx*= 1.5
Paso 5.4: Si x<90). marc_p2 += 1;
Paso 5.5: Si x<805). marc_p2 += 1;
Paso 6: Se llama a la función void *cancha(void *args)
Paso 6.1: Se declara un calor de tipo entero r=15.
Paso 6.1.1: Se declara el color del contorno de la cancha (WHITE).
Paso 6.2: Mientras 1==1.
Paso 6.2.1: Se declara la función rectangle, con argumentos (XMin-(r+1),YMin-(r+1),XMax+(r+1),YMax+(r+1).
Paso 7: Se llama a la función void *barras(void *args)
Paso 7.1: Se declara una función de tipo entero (int click)= 0.
Paso 7.1.1: Se declara la función rectangle (85,coordx1,95,coordy1), para el tamaño y posición de la barra de la derecha.
Paso 7.1.2: Se declara la función rectangle(800,coordy2,810,coordx2); para el tamaño y posición de la barra de la izquierda.
Paso 7.2: Mientras 1==1.
Paso 7.2.1: Click es igual a getch.
Paso 7.3: Se declara un menú con argumento click y se derivan 4 casos.
Paso 7.3.1: Case 1, se declara la letra en código ASCII para mover la barra S-(115).
Paso 7.3.1.1: Si (coordx1 <= YMax). coordy1 += 10, coordx1 +=10.
Paso 7.3.1.2: Se limpia pantalla.
Paso 7.3.1.3: Se manda a llamar la función rectangle(85,coordx1,95,coordy1), rectangle(800,coordy2,810,coordx2);
Paso 7.3.1.4: Se rompe.
Paso 7.3.2: Case 2, se declara la letra en código ASCII para mover la barra W-(119).
Paso 7.3.2.1: Si (coordy1 >= YMin). coordy1 -= 10, coordx1 -=10.
Paso 7.3.2.2: Se limpia pantalla.
Paso 7.3.2.3: Se manda a llamar la función rectangle(85,coordx1,95,coordy1), rectangle(800,coordy2,810,coordx2);
Paso 7.3.2.4: Se rompe.

Paso 7.3.3: Case 3, se declara la letra en código ASCII para mover la barra L-(108).
Paso 7.3.3.1: Si (coordx2 <= YMax). coordy2 += 10, coordx2 +=10.
Paso 7.3.3.2: Se limpia pantalla.
Paso 7.3.3.3: Se manda a llamar la función rectangle(85,coordx1,95,coordy1), rectangle(800,coordy2,810,coordx2);
Paso 7.3.3.4: Se rompe.
Paso 7.3.4: Case 4, se declara la letra en código ASCII para mover la barra O-(111).
Paso 7.3.4.1: Si (coordy2 <= YMin). Coordy2 -= 10, coordx2 -=10.
Paso 7.3.4.2: Se limpia pantalla.
Paso 7.3.4.3: Se manda a llamar la función rectangle(85,coordx1,95,coordy1), rectangle(800,coordy2,810,coordx2);
Paso 7.3.4.4: Se rompe.
Paso 8: Se manda a llamar la función void Init()
Paso 8.1: Se declara las varibles inciailes, estas están por defecto, por tanto, no se pueden modificar. “int gd = DETECT, gm”
Paso 8.2: Se declara el auxiliar de concola, donde se dibujará el videojuego. **“**initgraph(&gd, &gm, "C:\\TC\\BGI");**”**
Paso 8.3: Se declara el tamaño de pantalla para la visualización del videojuego. “initwindow(1080,720);”

Conclusiones

Díaz Matus Ricardo: En conclusión es que en este proyecto fue divertido realizarlo, pero también algo difícil, ya que tuve algunas dificultades para poder implantar el movimiento de la barra en gráficos, al igual de poder incluirla dentro de la cancha, junto con la pelota, pero con la colaboración de equipo que tuve con mi compañero, pudimos resolver, mediante una estructura llamada hilos en Lenguaje C, las cual nos ayudó a que funcionará el videojuego y además cumpliera nuestras expectativas. Pero aun así tengo algunos problemas en entender y plantear los problemas al lenguaje C, es decir, tengo la idea de cómo resolver los problemas de manera analítica, pero para pasar mi idea al lenguaje máquina, me cuesta un poco entender como estructurarlos. Sin embargo espero yo que con la práctica de más programas esto se vaya minorando poco a poco.

Martínez Olivares Vicente Jafet: En conclusión llevar a cabo el uso de todo lo aprendido durante el curso y plasmarlo en un proyecto, que en este caso fuel la realización de un videojuego llamado “PONG”, nos permite tener un panorama muy abierto sobre las aplicaciones que tiene en programar en Lenguaje C, ya que también este lenguaje, que es considerado de bajo nivel, tiene sus defectos, ya que la sintaxis en estructuras particularmente hablando es algo difícil de comprender y por ende tiende a ser complicado a la hora de ejecutar analizar un algoritmo, para convertirlo a código, para solucionar dicho problema planteado. En lo que concierne al videojuego, se llegó a complicar un poco, ya que teníamos el problema de como poder juntar las barras junto a la cancha y la pelota y a la vez mover dichas barras sin que se cerrará el compilador.

Pero con la ayuda de mi compañero pudimos solucionarlo con el apoyo de una estructura llamado hilos en lenguaje C.

Anexos:

Conceptos de Hilo o hebra

Un proceso típico de Unix puede ser visto como un único **hilo de control**: cada proceso hace sólo una cosa a la vez. Con múltiples **hilos de control** podemos hace más de una cosa a la vez cuando cada hilo de hace cargo de una tarea. Beneficios de hilos:

- Se puede manejar eventos asíncronos asignando un hilo a cada tipo de evento. Luego cada hilo maneja sus eventos en forma sincrónica.
- Los hilos de un proceso comparten el mismo espacio de direcciones y descriptores de archivos.
- Procesos con múltiples tareas independientes pueden terminar antes si estas tareas se desarrollan traslapadamente en hilos separados. De este modo tiempos de espera de la primera tarea no retrasan la segunda.
- Programas interactivos pueden lograr mejor tiempo de respuesta usando hilos para manejar la entrada y salida. Este es un ejemplo del punto previo.
- La creación de un hilo es mucho más rápida y toma menos recursos que la creación de un proceso.

Administración de Hilos

Un paquete de manejo de hilos generalmente incluye funciones para: crear y destruir un hilo, iteración, forzar exclusión mutua y espera condicionada.
Los hilos de un proceso comparten variables globales, descriptores de archivos abiertos, y pueden cooperar o interferir con otros hilos.
Todas las funciones de hilos del POSIX comienzan con pthread. Entre ellas están:

Función POSIX	Descripción (páginas man, otra)
pthread_equal	verifica igualdad de dos identificados de hilos
pthread_self	retorna ID de propio hilo (análogo a getpid)
pthread_create	crea un hilo (análogo a fork)
pthread_exit	termina el hilo sin terminar el proceso (análogo a exit)
pthread_join	espera por el término de un hilo (análogo a waitpid)
pthread_cancel	Termina otro hilo (análogo a abort)
pthread_detach	Configura liberación de recursos cuando termina
pthread_kill	envía una señal a un hilo

Creación de Hilos

Los procesos normalmente corren como un hilo único. La creación de un nuevo hilo se logra vía pthread_create.

# include <pthread.h> int **pthread_create**(pthread_t * restrict tidp, const pthread_attr_t * restrict attr, void * (* start_routine) (void *), void * restrict arg);	tidp: salida, puntero a id del hilo attr: entrada, para definir atributos del hilo, null para default start_routine: entrada, función a correr por el hilo arg: entrada, argumento de la función del hilo. La función debe retornar un * void, el cual es interpretado como el estatus de término por pthread_join

Término de un Hilo

Si un hilo invoca a exit, _Exit o _exit, todo el proceso terminará.

Un hilo puede terminar de tres maneras sin terminar el proceso: Retornando de su rutina de inicio, cancelado por otro hilo del mismo proceso, o llamando pthread_exit.

#include <pthread.h> void **pthread_exit** (void * rval_ptr);	rval_ptr queda disponible para otros hilos al llamar pthread_join rval_ptr debe existir después del término del hilo.
int **pthread_join**(pthread_t tid, void ** rval_ptr);	El hilo llamante se bloquea hasta el término del hilo indicado. Si el hilo en cuestión es cancelado, rval_prt toma el valor PTHREAD_CANCELED Si no estamos interesados en el valor retornado, poner NULL.
int **pthread_cancel**(pthread_t tid);	Permite a un hilo cancelar otro hilo del mismo proceso. Retorno 0 es OK, !=0 => error. Equivale a si el hilo indicado llamara pthread_exit(PTHREAD_CANCELED); sin embargo, un hilo puede ignorar este requerimiento o controlar cómo se cancela. pthread_cancel sólo hace un requerimiento, pero no lo fuerza.
int **pthread_setcacelstate**(int state, int * oldstate);	Permite cambiar el estado del hilo a PTHREAD_CANCEL_ENABLE (default) o PTHREAD_CANCEL_DISABLE, en este estado el hilo ignora llamados a pthread_cancel que le afecten.