Universidad Nacional Autónoma de México

Facultad de Ingeniería

Asignatura: Estructura y Programación de Computadoras

PROYECTO FINAL: Juego "Galaga"

Profesor: M.I. Luis Sergio Durán Arenas
Semestre 2024-1

Equipo: 9

Grupo: 2

Integrantes:
- Fernández Cano Iván Antonio
- Flores Rivera Brenda Lucrecia
- Ramírez Monzón Ana Cristina
- Troncoso González Carlos Andrés

Introducción

Con este proyecto, se busca generar un programa que sea completamente funcional, simulando un videojuego conocido como “Galaga”, donde el usuario tiene varias opciones en pantalla, mediante botones se representan las funciones de iniciar el juego, pausar el juego y de reiniciar el juego.

Al presionar el botón para iniciar el juego, se debe comenzar a mover la nave enemiga de forma automática a los lados y hacia abajo (en referencia a la nave del jugador), también debe emitir disparos que afecten a la nave del jugador, cada que se provoque un impacto mediante una bala, se debe restar una vida al jugador, disminuyendo la cantidad de caracteres que representan las vidas restantes.

El jugador también podrá moverse, pero únicamente a los lados (respetando el límite de la pantalla visible del juego, tanto por parte del jugador, como del enemigo), la nave del jugador también debe ser capaz de disparar, si uno de los disparos llega a impactar con la nave enemiga, ésta recibirá cierto daño, que al llegar a cierta cantidad, provocará la desaparición de la nave y la reaparición de otra en su lugar, aumentando la puntuación del jugador por cada impacto.

En cada ejecución del programa se debe llevar un registro de la máxima puntuación alcanzada, esto se muestra en pantalla mediante la highscore, que se actualiza si al terminar cada juego, la puntuación registrada es mayor que la highscore registrada anteriormente.

El botón de pausa debe forzar al programa a detenerse en la posición en que se encuentra actualmente, y en el botón de reinicio, se debe borrar absolutamente todo y reiniciar la puntuación y ubicación de ambas naves.

Lo que se debe hacer para la resolución de la problemática, es agregar las funciones necesarias al programa para poder realizar el movimiento de la nave del jugador, además de generar cierto patrón de movimiento y disparo para la nave enemiga, teniendo en cuenta la posición del carácter que representa la bala para tomar en cuenta dentro de la puntuación del jugador o el daño hecho al enemigo.

El programa debe presentar varios flujos para que dé la apariencia de que se están ejecutando varias tareas al mismo tiempo, y no hayan interrupciones entre los procesos, esto se logra mediante la selección que ya hemos revisado en las clases de la asignatura.

Desarrollo

Para la resolución de la problemática, la brigada comenzó con una base que representaba la interfaz gráfica del juego, posteriormente se fueron agregando procedimientos y etiquetas que separan e indican cada procedimiento a realizar.

Se buscó generar un efecto de simultaneidad entre las operaciones que se llevan a cabo mediante el uso de comparaciones y saltos condicionales, así como el uso de nuevos procedimientos y lógicas que son necesarias para la implementación del juego en sí.

Flujo principal del programa

Funcionamiento de los botones.

Después de dibujar la base de la interfaz gráfica y realizar la conversión del mouse para la resolución de la ventana actual, se verifica primero si se ha presionado el click izquierdo del mouse en alguno de los botones disponibles del programa. La verificación de los botones se hace mediante los registros CX (columna del cursor) y DX (renglòn del cursor), por lo que verificamos la posición del cursor mediante saltos etiquetas distintas para concluir si se presionó en el área que abarca el botón.

Un detalle importante a considerar es la etiqueta '**testplay**', aquí se salta en caso de que no se haya presionado ninguna tecla del mouse, además, usamos una variable booleana implementada que sirve para saber el estado del programa:

- 1: se encuentra en ejecución (es decir, se presionó el botón play).
- 0: no está en ejecución.

Existe otra variable booleana llamada 'boolGameOver', la cuál es la encargada de verificar si se debe dibujar en pantalla un mensaje de que el juego ha terminado o no.

El **botón de play**, si fue presionado, se encarga de cambiar el estado de las dos variables booleanas anteriormente mencionadas, indicando así que el juego están en ejecución (bool_play es 1) y que no se muestra en pantalla el mensaje 'Game over' ('boolGameOver' es 0). Cuando se presiona este botón, se salta a la etiqueta 'play' para empezar con la lógica en sí del juego.

El botón de pausa, cuando es presionado, se cambia el estado de la variable booleana que estamos utilizando como indicador (bool_play) , su valor se actualiza a ser cero, y posteriormente se redirige a la etiqueta de mouse_no_clic, donde se detendrán los procesos al detectar el valor del indicador.

El botón de reinicio, cuando es presionado, se redirige el flujo del programa a la etiqueta "reiniciar", donde se reduce el número de vidas del jugador a 1 y se usa la función que indica que la nave ha recibido un impacto ,para que automáticamente el

número de vidas sea cero, entonces se detecta este cambio, y se muestra la pantalla de “game over”, reiniciando el número de vidas, las posiciones de ambas naves, y actualizando la highscore si es necesario.

El salto hacia ‘mouse_no_clic’ nos permite verificar constantemente si algún botón fue presionado a partir del mouse.

Control de la nave blanca (usuario)

Una vez que nos encontramos en la etiqueta ‘play’, utilizamos dos diferentes opciones de la interrupción 16h (en ese orden):

- 01h: comprobamos el buffer del teclado, para que en caso de que se detecte que no hay ninguna tecla presionada, comiencen las acciones de la nave enemiga.
- 00h: si se presionó una tecla, entonces revisamos a ver si es alguna de las teclas con las que el usuario puede interactuar para mover o disparar su nave.

Como la entrada del teclado se guarda en el registro ‘AL’, entonces se hace una comparación por cada tecla con su valor ascii, si coincide con su respectivo valor, se realiza alguna de las acciones de movimiento o se disparan las balas de la nave.

En este punto, es importante mencionar la lógica para que la nave enemiga dispare. Esto se logra con la implementación de un contador que se incrementa cada vez que el usuario presiona una tecla (independientemente si se realiza alguna acción con ella o no). Cada vez que este contador llega a un valor de 20, es ahí cuando comienza el proceso para disparar una bala de la nave enemiga, logrando así un comportamiento más ordenado por parte de las balas enemigas.

Por la razón anterior es que en cada etiqueta donde se realiza alguna acción del usuario, finalmente se termina saltando a la etiqueta ‘llamar_disparo_enemigo’, ya que se verifica el valor del contador para saber si se tiene que disparar una bala enemiga o no.

Los procedimientos que realizan algún efecto directo en el control de la nave blanca son los siguientes:

- **MOVER_IZQUIERDA, MOVER_DERECHA**

Ambos procedimientos decrementan o incrementan la columna actual del jugador, dependiendo si se trate de un movimiento a la izquierda o a la derecha. Primero se borra la nave en pantalla, se modifica la columna y posteriormente se vuelve a imprimir en la nueva posición.

Para delimitar el movimiento de la nave blanca y que no rebase el área de juego delimitada por la interfaz gráfica, se verifican en las etiquetas ‘flecha_izquierda’ y ‘flecha_derecha’ ubicadas dentro de la etiqueta play. Lo anterior, permite limitar el movimiento de la nave blanca para que no se ejecute la llamada a los procedimientos de movimiento anteriores.

- player_col = 4: límite para el movimiento a la izquierda.
- player_col = 36: límite para el movimiento a la derecha.

- **DISPARAR_BALA_AMIGA**

Las balas de la nuestra nave se almacenan en tres arreglos de longitud cuatro, por lo que esta es la mayor cantidad de balas blancas que pueden estar en pantalla al mismo tiempo (en el caso de las rojas es igual, pero en realidad es muy difícil conseguir que haya más de una bala enemiga en pantalla).

El primer arreglo es booleano, sirve para determinar si esa bala está o no en ejecución, los otros dos arreglos sirven para almacenar la posición y renglón en el que se encuentran.

Lo que hace este procedimiento es buscar en el arreglo booleano una posición “disponible” para inicializar la ejecución de una bala, si no encuentra una disponible, simplemente no se dispara nada.

Si encuentra una posición disponible (valor 0 en el arreglo booleano), cambia ese valor a uno, e inicializa el valor del índice de esa bala en los otros dos arreglos con la posición de la punta de la nave blanca.

- **AVANZAR_BALAS_AMIGAS**

Para simular que el programa es paralelo, lo que sucede es que al final de cada ciclo de ejecución del juego, se aumenta un contador (y después se repite el ciclo). Si el contador alcanza el valor FFFh, se avanzan tanto las balas amigas como las enemigas. **En otras palabras, las balas en pantalla van a la velocidad a la que el programa ejecuta 0FFFh ciclos de “play” del juego.**

Este procedimiento comienza recorriendo el arreglo booleano donde se almacenan las balas que están en ejecución; al encontrar una de ellas (1 en el arreglo booleano), se borra la bala de la pantalla y se avanza. Antes de imprimir la bala en su nueva posición, se revisa si en esa posición hay un carácter del color y con fondo del color con el que se dibuja la nave roja, determinando si impactó; en cuyo caso

se llama al método correspondiente y se termina la ejecución de la bala. En caso contrario, se verifica si la bala llegó a la última posición en pantalla, en cuyo caso su ejecución también termina. Si ninguna de estas dos cosas sucedió, simplemente se imprime la bala en su nueva posición. **La forma en la que una bala termina su ejecución, es que ya no se imprime, y su valor en el arreglo booleano cambia a 0.**

Concurrencia en el desarrollo del proyecto

Al principio, se presentó la complicación de cómo es que íbamos a juntar todas las funcionalidades tanto de la nave blanca como roja para que diera la impresión al usuario de que todo se está ejecutando al mismo tiempo. Esto al principio nos aterrizó a la idea del manejo de hilos pero en el proyecto se implementó una estrategia para darle una concurrencia a todos los eventos que fueran sucediendo en la ejecución del programa.

Se trata del uso de variables que actúan como contadores para realizar el movimiento de la nave enemiga y el movimiento de las balas de ambas naves, como se muestra en la siguiente imagen:

```
492  llamar_enemigo:
493      inc [contador]
494      cmp [contador], 1FFFh
495      jne llamar_mov_balas
496      mov [contador], 0
497
498      call MOVER_ENEMIGO
499
500  llamar_mov_balas:
501      inc [conta_mov_bala]
502      cmp [conta_mov_bala], 0FFFh
503      jne mouse
504      mov [conta_mov_bala],0
505      call AVANZAR_BALAS_AMIGAS
506      call AVANZAR_BALAS_ENEMIGAS
507      jmp mouse
```

Antes de llamar al procedimiento que ejecuta el movimiento de la nave roja, se incrementa un contador y se compara con un valor considerablemente grande (1FFFh), solo en caso de que dicho contador alcance ese límite es entonces cuando se llama a MOVER ENEMIGO. La misma idea se plasma en la segunda etiqueta que realiza el movimiento de las balas de ambas naves.

Como la computadora realiza dichas operaciones (incremento y comparación) muy rápido, aprovechamos eso para gestionar a la nave enemiga, de tal manera que como al final se termina regresando con un salto a la etiqueta mouse, permite generar la sensación de que el usuario está controlando la nave blanca al mismo

tiempo que se ejecutan los movimientos de la nave roja y de las balas; además de que se puedan presionar los botones y que su acción parezca inmediata.

La razón de ser de los valores 1FFFh y 0FFFh en ambos contadores, tiene un efecto directo con la velocidad en la que se va realizar dicho movimiento, ya que entre más grande sean estos límites, entonces se tardará más en ejecutar las acciones de movimiento, por lo que finalmente decidimos que estos valores eran los idóneos para una sensación buena de juego.

Una vez que se alcanza el límite establecido, se reinicia el contador para que se vuelva a incrementar y comparar sucesivamente durante la ejecución del programa.

Manejo de las acciones de la nave enemiga

Lo primero que se verifica es el contador antes mencionado, justamente para ver si es el momento en el que la nave enemiga dispara. En dado caso de que no, entonces se manda primero a llamar al movimiento de la nave, con la estrategia antes mencionada para brindar la concurrencia al programa. Cuando termina el procedimiento del movimiento, entramos en la etiqueta 'llamar_mov_balas', que realiza las llamadas para el movimiento de los disparos de ambas naves.

Cuando conta_enemigo llega a 20 (es decir, se presionaron 20 teclas en total), saltamos a la etiqueta 'dispara enemigo' para reiniciar el contador y llamar al procedimiento que dispara las balas de la nave enemiga.

Se implementaron los siguientes procedimientos relacionados con las acciones de la nave enemiga:

- **MOVER_ENEMIGO**

Este es llamado para mover a la nave enemiga dependiendo de la cantidad de vidas que tenga el usuario, por lo que primero investigamos la cantidad de vidas actual que posee el usuario para después llamar al procedimiento indicado para mover a la nave roja.

Al igual que para la nave del usuario, se agregaron 4 procedimientos similares a los correspondientes de la nave blanca:

- ENEMIGO_IZQUIERDA y ENEMIGO_DERECHA.
- ENEMIGO_ABAJO Y ENEMIGO_ARRIBA.

Los 4 anteriores procedimientos incrementan o decrementan la columna del enemigo (para el movimiento horizontal) y el renglón de este (movimiento vertical).

En total se implementaron 3 patrones de movimiento, configurados manualmente teniendo en cuenta las actualizaciones que se iban presentando en las variables ‘enemy_col’ y ‘enemy_ren’.

Los procedimientos **PATRON3, PATRON2, PATRON1** tienen una lógica similar y solo varían específicamente en los movimientos que se presentan en la nave; el patrón que se ejecuta va acorde a la cantidad de vidas que tiene el jugador en ese momento.

La estrategia a seguir fue primero diseñar el movimiento de la nave por a parte a través de solamente el uso de etiquetas y, posteriormente, asignar un número empezando desde el 1 hasta el número total de etiquetas que tuviera dicho patrón, en otras palabras, una especie de switch-case para saber qué parte en específico del movimiento ejecutar.

Recordando que las variables declaradas en el segmento de datos se consideran globales, aprovechamos eso para diseñar 3 variables en específico para cada patrón que nos indican el movimiento exacto que se debe de ejecutar en ese momento del patrón. Después de llamar a algún procedimiento que afecte a la posición de la nave enemiga, para no perder la concurrencia que tenemos en nuestro programa, se almacena en la variable ‘var_patron#’ el movimiento siguiente a ejecutar en ese patrón de la nave; para que en la siguiente ocasión que se llame al procedimiento de un patrón en específico, salte a la etiqueta correspondiente gracias al “switch-case” del principio de este procedimiento.

Nota: en ‘var_patron#’, el símbolo # representa el número correspondiente al patrón que se está ejecutando (1, 2, ó 3).

De esta manera, el programa puede “recordar” en qué parte del patrón de movimiento de la nave enemiga se queda cada vez que la mueve, de forma que permite pasar a ejecutar otras cosas antes de volver a mover la nave..

- **DISPARAR_BALA_ENEMIGA**

Este método hace lo mismo que DISPARAR_BALA_AMIGA pero usando sus tres arreglos booleanos para balas enemigas, e inicializando la bala en la posición de la punta de la nave enemiga.

- **AVANZAR_BALAS_ENEMIGAS**

Funciona exactamente igual que el método para las balas amigas, pero moviendo las balas en dirección contraria, verificando si impactan en la nave blanca o en el fin de la pantalla.

Otros procedimientos que se ejecutan cuando suceden ciertos eventos.

- **IMPACTAR_BLANCA**

  Decrementa la cantidad de vidas del usuario, si después de esto aún le quedan, borra las vidas de la UI e imprime la nueva cantidad; posteriormente regresa ambas naves a su posición inicial.

  En el caso de que la nave blanca ya no tenga más vidas, se reinician a 3, se actualiza el high score si fue superado, se “enciende” la variable booleana de “Game Over”, y se “apaga” la variable booleana que indica si el juego está en ejecución. En dado caso, el procedimiento obliga a que el código regrese a ejecutarse desde “inicio”.

- **IMPACTAR_ROJA**

  Incrementa un contador de los impactos que se le han dado a esa nave roja, se incrementa el score (se borra y se imprime el nuevo); y posteriormente se compara el contador de impactos con 3. Si a la nave ya se le han dado 3 impactos, se “muere”; de manera que se borra, y “aparece” una nueva en la posición inicial. Esta nueva nave empieza con su contador de impactos nuevamente en 0.

- **PRINT_ENEMY**

  (MODIFICACIÓN PARA VERIFICAR COLISIÓN ENTRE NAVES){

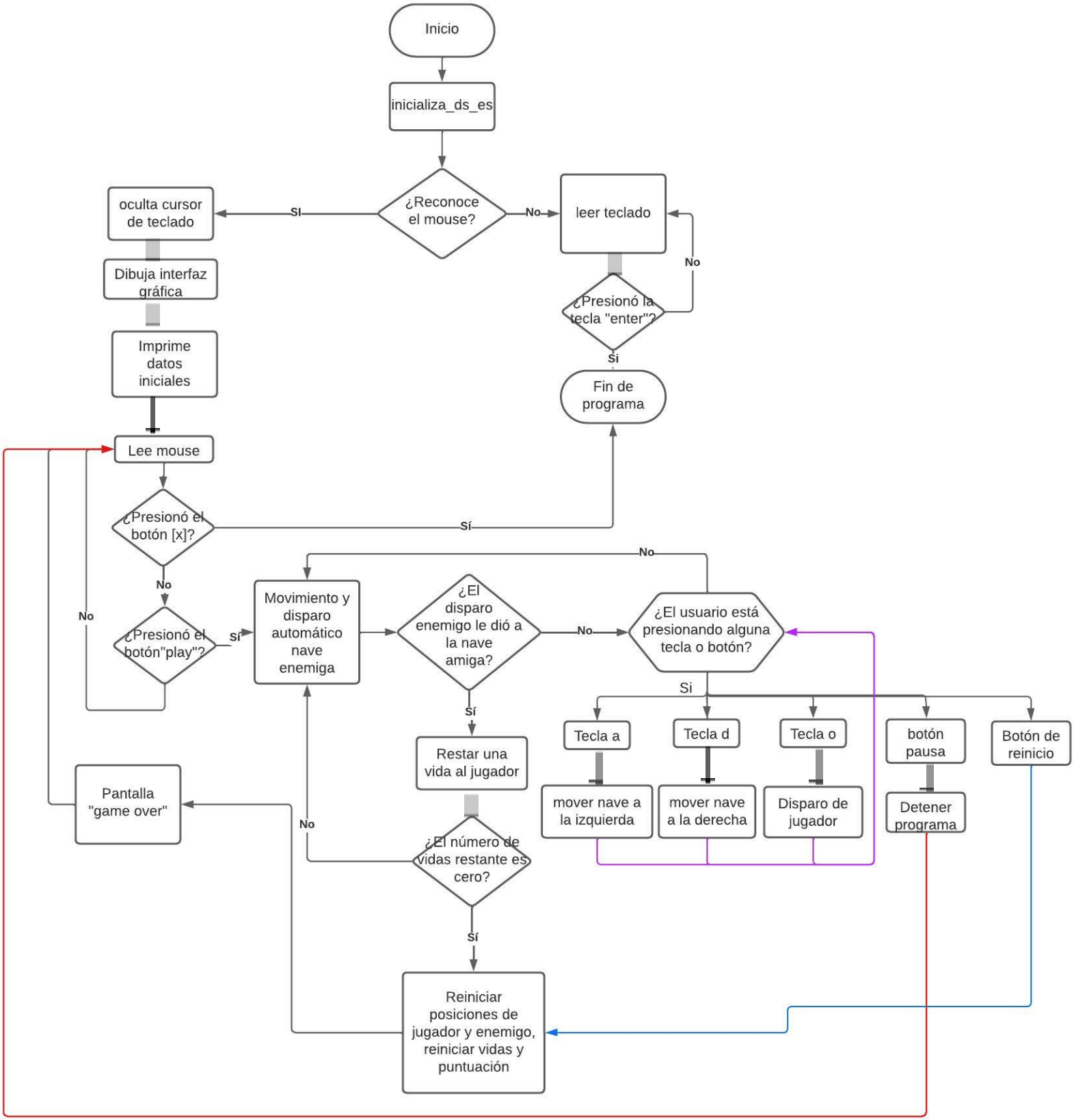
  A este método le modificamos lo que pasa cuando se va a imprimir la punta de la nave roja, ya que ahora **actúa como un sensor de colisión con la nave blanca**. Antes de imprimirse, verifica si en esa posición hay un caracter de los que conforman a la nave blanca, en cuyo caso invoca el método de IMPACTAR_BLANCA, y después de esto detendría su ejecución. En caso contrario, el método actúa de la forma tradicional.

Pantalla de Game Over cuando se acaban las vidas

En el segmento de datos, se declararon un total de 8 cadenas específicas para la impresión de un mensaje de ‘Game Over’ una vez que el usuario agota sus vidas totales. Su impresión en pantalla se logra con ayuda de una variable booleana llamada ‘bool_GameOver’, en donde justo después de la etiqueta imprime_ui al inicio del flujo principal del programa, se verifica si se debe imprimir dicho mensaje en pantalla o no. Un valor de 1 indica que sí, por lo que llamada a su método correspondiente para dibujarlo, en dado caso de que no, esa impresión se la salta.

Cabe recalcar que solamente se muestra este mensaje en pantalla una vez que el usuario pierde, específicamente en la etiqueta ‘IMPACTAR_BLANCA’. De igual manera, el mensaje en pantalla se borra cuando el usuario vuelve a presionar el botón de play, dando a entender que quiere empezar una nueva partida y por ende se limpia ese mismo apartado de la misma.

Diagrama de flujo:



Conclusiones

- Fernández Cano Iván Antonio:

Durante el desarrollo de este proyecto, abarqué conceptos fundamentales que abarcan desde las bases de la programación de lenguaje de ensamblador hasta la lógica en sí en programación. Creo que es un buen desafío teniendo en cuenta todos los registros que repasamos a lo largo de todo el curso, ya que utilizamos el manejo de registros, direccionamiento de estos y otras cuestiones como el manejo de eventos de usuario en donde con una buena planificación por parte de todo el equipo y un trabajo constante, fue posible elaborar este juego. En general no sólo siento que apliqué bien los conocimientos acerca de lenguaje ensamblador, sino que también me ayudó a seccionar mi código de tal manera que tuviera procedimientos bien definidos y claros, también el hecho de que no es tan fácil de aprender a diferencia de otros lenguajes, me ayudó a pensar de una manera distinta la resolución de problemas en cuanto a la lógica de programación se refiere.
Más que complicado, creo que fue un proyecto en donde teníamos que tener bien definida la estructura de cómo iba a ser desde un principio, es por ello que creo que como equipo tuvimos una buena organización y planificación de las funcionalidades del programa. Además, los problemas que fueron surgiendo en la elaboración de este como la concurrencia en el juego, el control del flujo, o incluso la lógica en sí del juego, se fueron tratando poco a poco y resolviendo a medida que íbamos avanzando en el programa. De verdad creo que se demuestra un buen manejo de los conceptos vistos a largo del curso por mi y por los integrantes del equipo.

- Flores Rivera Brenda Lucrecia:

Durante el desarrollo de este proyecto, se abordó a fondo la programación en lenguaje ensamblador. El manejo de registros, su direccionamiento y la gestión de eventos de usuario demandaron una planificación minuciosa y un esfuerzo constante del equipo para elaborar el juego. No solo se aplicaron los conocimientos de lenguaje ensamblador, sino que también se segmentó el código en procedimientos definidos y claros, adaptándome a la complejidad de este lenguaje, lo que implicó un enfoque diferente al que estoy acostumbrada.
La estructura definida desde el principio requirió organización y planificación en la conceptualización y funcionalidades del programa. A medida que avanzábamos, enfrentamos desafíos como la concurrencia en el juego, el control del flujo y la lógica del mismo. Sin embargo, tanto yo como mis compañeros demostramos un sólido dominio de los conceptos aprendidos.
El análisis inicial del código del profesor nos permitió comprender su funcionamiento y estructura, identificando las funciones y elementos a agregar. Durante el desarrollo, utilizamos saltos, operaciones aritméticas y lógicas para mantener un flujo constante.

- Ramírez Monzón Ana Cristina:

Con la elaboración de éste proyecto, se pudo aplicar todo lo visto durante las clases; la brigada comenzó por analizar el código proporcionado por el profesor en un inicio para la mejor comprensión de su funcionamiento y estructura, para posteriormente separar e identificar las funciones y elementos que debíamos agregar, durante el desarrollo de las diferentes funciones, se hizo uso de varios elementos revisados durante el semestre, tales como el uso de saltos, operaciones aritméticas, operaciones lógicas, siempre teniendo en cuenta el flujo del programa para poder lograr que el juego pueda realizar varias operaciones y revisiones de forma constante, logrando así que el programa no se detenga mientras se termina de ejecutar alguna instrucción, también durante la escritura del programa se hizo uso de varios arreglos para almacenar información que permitiera el uso de varias animaciones de disparos en la pantalla, dando más libertad al jugador para atacar a la nave enemiga, también se generaron varios elementos extra a lo solicitado mediante el uso de varios patrones de movimiento para la nave enemiga, que continúan por ejecutarse a pesar de la reaparición de la nave enemiga, también se está revisando constantemente si el usuario se encuentra presionando la tecla que indica movimiento a la derecha o izquierda o disparo, además del área delimitada tanto para el movimiento de ambas naves como para la trayectoria de los disparos, algo muy importante a remarcar sobre este proyecto, es que motivó a que el equipo retomara lo visto en clase, y además investigara cómo usar ciertas funciones para lograr el efecto deseado, los diferentes puntos de vista proporcionados por mis compañeros fueron de mucha utilidad, ya que la gran parte del código que yo realicé presentó ciertas fallas o interrupciones durante la ejecución, sin embargo, con ayuda del equipo es que me di cuenta de mis errores, obteniendo mucho aprendizaje.

- Troncoso Gonzáles Carlos Andrés

Con la elaboración de este proyecto me fue posible aplicar todos los conceptos que fueron revisados durante el segundo tema del curso. Ya que en su implementación, diseñamos, creamos y utilizamos: variables, constantes, macros, bloques de código separados por etiquetas, procedimientos, operaciones sobre la pila, etc.

Durante el desarrollo del proyecto atravesé varias problemáticas intrínsecamente relacionadas con el uso de un lenguaje de bajo nivel, ya que hubieran sido fácilmente solucionadas si se usara un lenguaje de alto nivel. La principal de ellas fue lograr que varios procesos estuvieran en ejecución simultánea sin hacer uso de hilos. Para esto, la solución conceptual es hacer que el programa sea concurrente, de manera que administre, alterne, y otorgue tiempo de ejecución a cada uno de los procesos; de esta forma, la solución es conceptualmente igual al trabajo que se

encarga de realizar el Scheduler de Java, o cualquier símil de un lenguaje de alto nivel, encargados de administrar los hilos.

Además, en la implementación de varias particularidades del proyecto, fue necesario hacer uso de diversas interrupciones (y sus opciones) que no fueron revisadas durante las clases; por lo que este proyecto amplió mis conocimientos en la programación con el lenguaje ensamblador intel x86.

Finalmente, es importante destacar que la realización en equipo del proyecto nos llevó a la necesidad de separar su desarrollo en etapas, cada una con diversas tareas realizables simultáneamente. De esta forma, el proyecto también me brindó experiencia en la planeación y calendarización de proyectos, así como la repartición de tareas y el trabajo grupal para combinar versiones que agregan avances en diferentes aspectos del proyecto.