Metodología de aprendizaje de Máquinas de Estados Finitos mediante videojuegos

M. Á. Gonzalez Rodriguez, miguel_gonzalezr@javeriana.edu.co, D. F. López Escobar , dfelipe_lopez@javeriana.edu.co, J. H. Avendaño López , jaavendano@javeriana.edu.co, C. A. Sanchez Diaz, ca-sanchezd@javeriana.edu.co

***Abstract*—Las máquinas de Estados Finitos (FSM, por sus siglas del inglés) son la base de gran variedad de algorítmos de computación cuyo funcionamiento esta dado por el estado actual, donde solo se puede estar en un estado en un tiempo específico. En el presente documento, se busca aplicar el diseño de un videojuego retro basado en Space Invaders cuyo funcionamiento este dado por máquinas de Mealy. Además el objetivo planteado es generar un modelo concurrente y escalable, es decir, funcionamiento en diferentes Sistemas Operativos (OS) y placas de desarrollo que aplica y afianza las temáticas cubiertas en el curso de Diseño de Sistemas con Procesadores de la Pontificia Universidad Javeriana.**

***Index Terms*—FSM, Videojuegos, aprendizaje, concurrencia, sistemas embebidos.**

Fig. 1: Paralelismo vs. Concurrencia [1]

I. Introducción

EL desarrollo de videojuegos debe considerar diversos procesos y tareas que deben ejecutarse de manera simultanea, estos incluyen lectura del control/mando (procesamiento de entradas), animaciones en pantalla, ejecución de sonida, lectura de datos en memoria, entre otros. Sin embargo, un procesador solo puede ejecutar una tarea en un momento dado, por lo cual, las tareas son ejecutadas en pequeños procesos que aparentan un paralelismo, este proceso recibe el nombre de concurrencia (figura 1).

Teniendo en mente lo anterior es necesario abstraer las diferentes partes del proceso y considerarlas pequeñas unidades que deben ser integradas sin afectar el procesamiento de las demás partes, para ello se considera la implementación de Máquinas de Estados Finitos, específicamente de Mealy, que se encargaran de las distintas tareas.

El caso propuesto es orientado al diseño de un juego de Space Invaders, para ello se consideran como entidades los extraterrestres (aliens), la nave (starship), las balas que puede pertenecer a los alien o a la nave, las barreras, el marco de juego y los indicadores/marcadores. Adicionalmente, se consideran tiempos independientes para cada uno de ellos y que no deben intervenir en otros procesos, por ejemplo, acaparando el procesador.

II. Diseño e interfaz

Una parte fundamental de los videojuegos resulta ser la visualización. Este aspecto no se centrará en gráficas de alta definición que caracterizan los videojuegos hiperrealistas de última generación, sino se desea orientar a la búsqueda de simplicidad en honor a los videojuegos clásicos de arcade. Para ello, se propone el uso de caracteres especiales de unicode [2] en el diseño.
En primer lugar, para las fronteras se emplea *unicode 2500* en referencia al diseño de cajas presentado en [3].
En segundo lugar, para la nave espacial se usan los elementos presentes en el *unicode 2580* referente a elementos de bloque extraido de [4].
En tercer lugar, para el diseño del alien se propone el uso de *unicode braile* debido a que se desean añadir múltiples instancias y los anteriores modelos son de tamaño notorio, esta resulta ser una opción viable en la implementación, además se consideran los modelos presentados en [5].
Para los indicadores y marcadores, también se propuso el uso de *unicode braile*. A diferencia de los aliens, el diseño fue elaborado de manera propia.
Por último, para la bala, se decide por el uso de caracteres estándar o relacionados a ASCII común.

Lo anterior es resumido en la figura 2 que muestra el prototipo considerado para la plantilla de juego.

Fig. 2: Diseño considerado para los objetos del juego

III. Implementación en código C

Para la codificación se plantean 4 máquinas de estado base, en referencia a alien, nave, bala y temporizador. Las cuales son presentadas en las figuras 3-6.

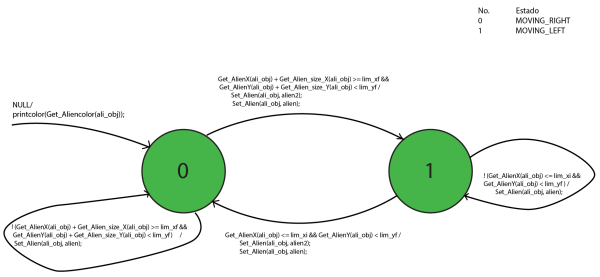

Fig. 3: FSM que describe el comportamiento de un alien.

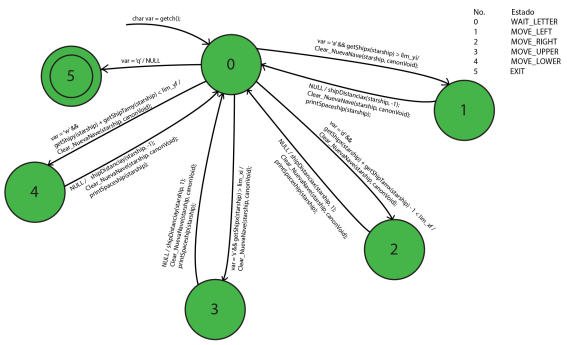

Fig. 4: FSM que describe el comportamiento de una nave.

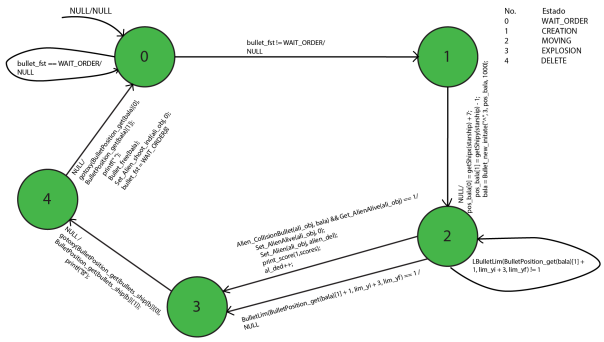

Fig. 5: FSM que describe el comportamiento de una bala.

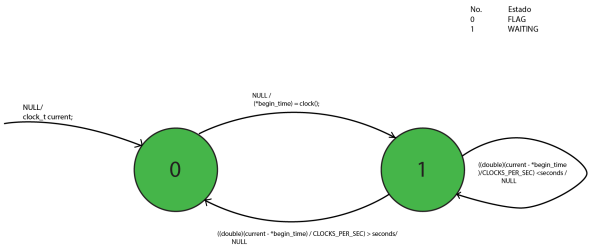

Fig. 6: FSM que describe el comportamiento de un temporizador

Se desea resaltar que la máquina de estados referente al temportizador será clave en el funcionamiento del programa, debido a que existirá una instancia por cada objeto en el programa, es decir, cada alien estará vinculado a un temporizador, y este regulará las acciones, donde su acción principal de acuerdo a la figura 6 es de bandera, sus estados oscilarán entre listo y en espera de acuerdo al tiempo asignado y el tiempo trasncurrido. Aquí el tiempo asignado se considera como atributo de todas las clases que deberá ser pasado por parámetro a este algorítmo.

Otro aspecto importante a mencionar es su funcionamiento en relación al sistema operativo. A nivel código de implementan dos funciones ***kbhit*** y ***getch*** que no son tomadas de la librería ***getch*** y resultan ser una adaptación propuesta por [6] para evitar interrupción y acaparamiento por teclado. Sin embargo, el funcionamiento de estas está condicionado a sistemas o distribuciones basadas en Linux, por lo cual, la primera limitación de este es su funcionamiento en OS diferentes a Windows. Adicionalmente, al implementar funciones relacionadas al reloj (***clock***) del sistema, su uso en un IDE online (Entorno de desarrollo integrado) como es el caso de Replit®debido a que el reloj dependerá de parámetros en referencia al internet y la conectividad por lo cual, el sistema tiende a saturarse por la velocidad a la que puede llegar.

IV. Resultados

Con base a las secciones anteriores, se consturye el sistema para realizar pruebas en dos sistemas WSL (Windows Subsys-

tem for Linux) con Ubuntu 20.04 y Ubuntu 20.04 nativo para realizar las pruebas del sistema, obteniendo las visualizaciones de las figuras 7 y 8 respectivamente.

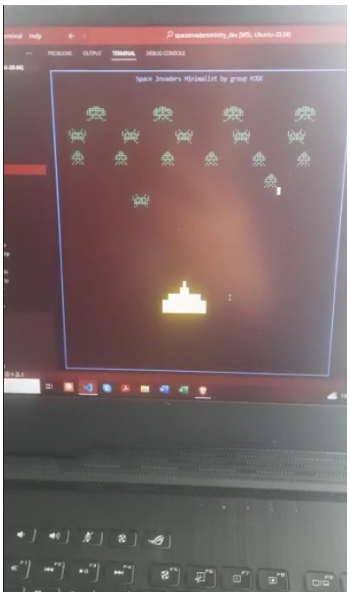

Fig. 7: Prueba realizada en PC con Windows y WSL

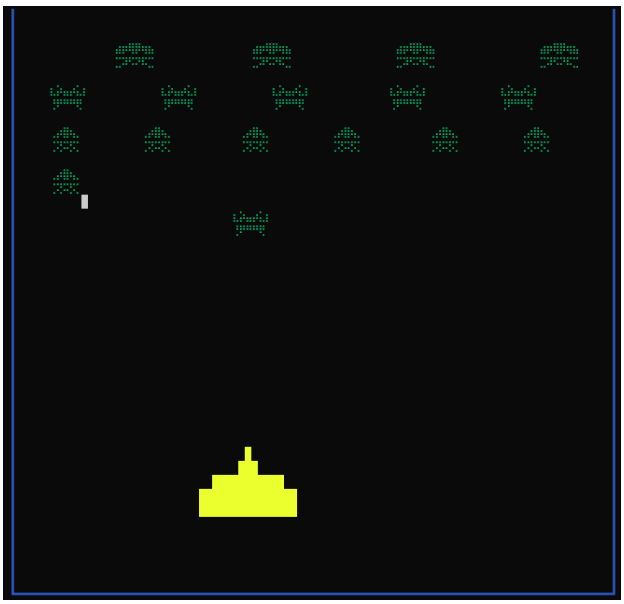

Fig. 8: Prueba realizada en PC con Ubuntu

Fig. 9: Prueba realizada en arcade casera con montaje en base a Raspberry Pi

Por un lado, para el caso de Ubuntu nativo, el programa funciona de manera estable con respuesta estable de las distintas entidades y manejo de teclas en tiempo real a estimulo sencillo, es decir, en caso de un solo pulso, dado que si se mantiene oprimido, la velocidad de respuesta del teclado se ve congelada debido a que queda manejando los casos acumulados dada la implementación de ***kbhit*** propuesta. Por el otro lado, el comportamiento en WSL es bastante similar con la excepcion de que este al incluir música y no estar directamente conectado a los periféricos se ve envuelto en una serie de excepciones manejables pero que afectan la gráfica del programa, por lo cual, para la versión de WSL se considera la desactivación de la música.

De igual forma, se propone una prueba en un sistema con menor capacidad, una raspberry pi 3 con Ubuntu instalado, No obstante, aquí se desea implementar la idea de un arcade. Luego de un remapeo de interruptores, es decir, la reconfiguración de teclado por botones y el cambio a esto, que permite una jugabilidad orientada al concepto original, la configuración resultante se presenta en la figura 9.

V. Discussion and Summary

Dadas las metodologías consideradas se valida el aprendizaje de FSM y concurrencia mediante proyectos didácticos que se presentan como un reto interesante para el estudiante y que lo motiva a profundizar en los aspectos de la clase e indigar más información para complementar el proceso. Además, se logra generar un diseño de software que cumple los objetivos, resaltando la escalabilidad en diferentes máquinas que sean configuradas con versiones o distribuciones de Windows.

A modo de sugerencia para trabajo posterior, se puede generar una variante de este proyecto orientada a ser portátil

y cuyo acceso sea mayor, implementando componentes más baratos, y cambiando la Raspberry por alguna tarjeta de desarrollo más sencilla, por ejemplo, Arduino.

ACKNOWLEDGMENT

Los autores desean expresar su agradecimiento a Juan Carlos Giraldo, profesor de la materia de Diseño de Sistemas con Procesadores en la Pontificia Universidad Javeriana por el apoyo a lo largo del proyecto y las orientaciones brindadas en la clase.

VI. REFERENCIAS

1 Free RTOS. (2020, Nov). MultiTasking, RTOS Fundamentals [Online]. Available: https://www.freertos.org/implementation/a00004.html
2 Unicode (1991). Unicode Charts[Online]. Available: https://unicode.org/charts/
3 Unicode (1991). Unicode Box Drawing [Online]. Available: https://unicode.org/charts/PDF/U2500.pdf
4 Unicode (1991). Unicode Blocks [Online]. Available: https://unicode.org/charts/PDF/U2580.pdf
5 macdice (2002, July 21) ASCII Invaders [Online] Available: https://github.com/macdice/ascii-invaders/blob/master/invaders.c
6 altinak (2009, Jan 15) Solution to C-non-blocking-keyboard-input [Online] Available: https://stackoverflow.com/questions/448944/c-non-blocking-keyboard-input