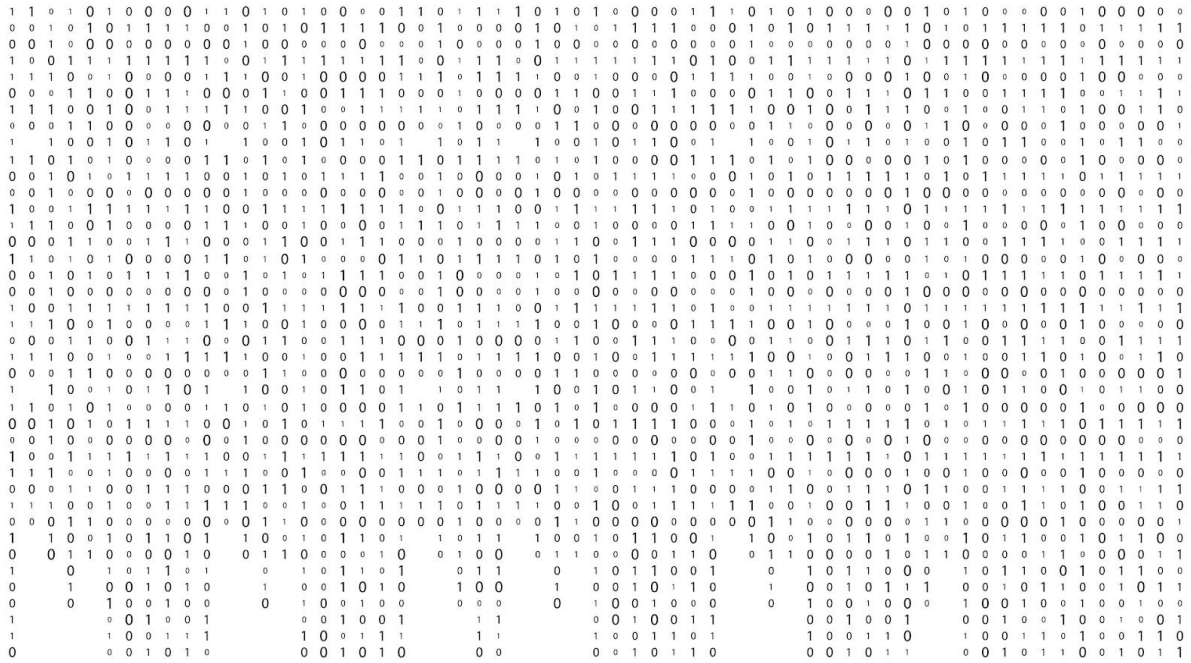
Universitat d'Alacant
Universidad de Alicante



MEMORIA DE PROYECTO RAZONAMIENTO AUTOMÁTICO VIDEOJUEGO PARA Z80 AREA 51 RAID

Juan Carlos López Gutiérrez
Antonio Aguirre Ivorra
Jorge Amil Pérez Ponsoda

Ficha Técnica

Tamaño Kbytes

- Area51Raid.cdt - 23.407 bytes (22,8KB)
- Area51Raid.cdt - 24.576 bytes (24,0KB)

Para el archivo binario el inicio se produce en la dirección de memoria 0x40 pero no es hasta la dirección 0x1177 donde se inicia el código, que llegará hasta 0x52C5.

Modo gráfico

El modo gráfico usado es el 0. Uso de doble buffer para el renderizado.

Detalles de implementación

Contenidos tecnológicos y diseños realizados

Arquitectura ES (simplificación Entidad - Componente - Sistema). Esta arquitectura, ampliamente utilizada en el desarrollo de videojuegos, distingue 3 elementos básicos.

- Componentes .- estructuras de datos
- Entidades .- conjuntos de componentes
- Sistemas .- procedimientos que transforman datos

En nuestro caso, hemos utilizado una estructura simplificada en la que no diferenciamos entidades, y todas ellas tienen los mismos componentes, aunque no los necesiten. Con esta decisión reducimos la complejidad pero incrementamos la cantidad de memoria usada.

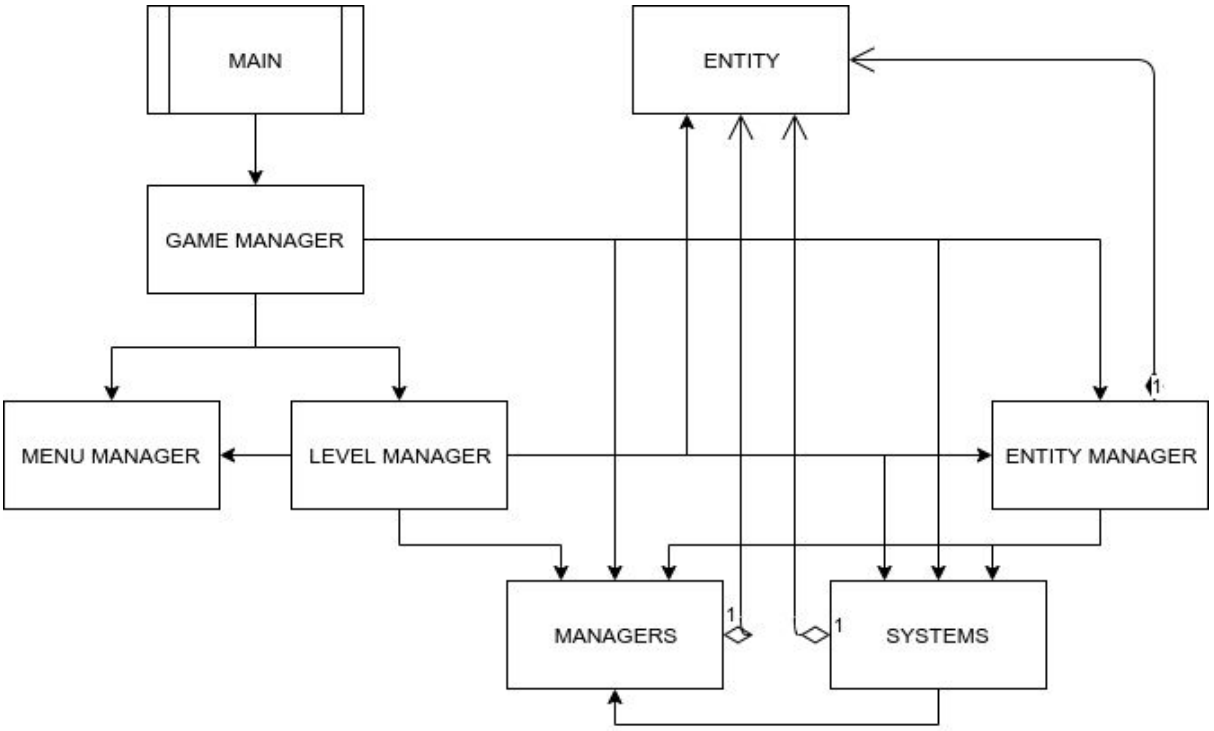
ECS antepone la composición a la herencia, evitando complejas jerarquías en las que es fácil caer en la programación orientada a objetos. En nuestro caso, si programando en lenguaje ensamblador de la misma manera que lo hacemos con lenguajes de alto nivel, rápidamente nos daríamos cuenta que nuestro código se ha vuelto inmanejable.

Gracias al principio de responsabilidad única podemos identificar una estructura clara en la que cada una de las partes tenga asignada sus propias tareas, facilitando el entendimiento de cómo funcionan o se interrelacionan las distintas partes entre sí.

Estructura Manager / System

La motivación de esta estructura también trata de separar responsabilidades, ampliando la arquitectura ECS. En este caso, cada manager se encarga de poseer y administrar los datos referentes a un componente concreto, mientras que los sistemas se encargan de transformar los datos de las entidades que requieran el uso de dicho componente. Esta estructura permite encapsular la gestión de los datos en los managers, los cuales darán acceso a los datos a sus respectivos sistemas.

Diagrama de clases


MAIN
ENTITY
GAME MANAGER
MENU MANAGER
LEVEL MANAGER
ENTITY MANAGER
MANAGERS
SYSTEMS
1
1
1
1

Optimizaciones y adaptaciones funcionales

Borrado de un elemento de un array

Para eliminar una entidad hemos implementado un método de borrado sencillo.

Dicho método recibe como parámetro de entrada, en el registro DE, el puntero a la entidad que debe eliminar, y destruye los registros HL, BC y AF.

Utilizando el registro HL como iterador sobre el array y BC como contador, vamos comprobando uno a uno cada puntero con el parámetro de entrada hasta el final del array.

Si encontramos una coincidencia, DE apunta al elemento a borrar (destino), HL al siguiente puntero (origen), BC contiene el número de elementos a copiar. De esta forma podemos utilizar la instrucción "ldir" para reestructurar el array y solo faltaria actualizar el puntero al último elemento del array.

Figura 1

```
; DE ptr to delete
entity_render_man_delete::
;; HL iter
    ld hl, #_render_ptr_array
    ld bc, #max_entities-1
_loop:
    push bc
;; BC ptr to check
    ld c, (hl)
    inc hl
    ld b, (hl)
    inc hl
;; if null END
    ld a, b
    or c
    ret z
;; BC == DE
    ld a, b
    cp d
    jr nz, _not_equal

    ld a, c
    cp e
    jr nz, _not_equal
```

Figura 2

```
_equal:
    ld d, h
    ld e, l

    dec de
    dec de

    pop bc

    push hl
    push de

    dec bc
    ld h, b
    ld l, c
    add hl, bc
    ld b, h
    ld c, l

    pop de
    pop hl

    ldir
```

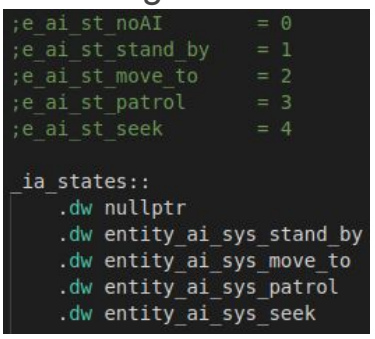
Figura 3

```
    ld hl, (_render_ptr_pend)
    dec hl
    dec hl
    ld (_render_ptr_pend), hl

    ret

_not_equal:
    pop bc
    dec bc
    jr _loop
```

Punteros a función

El sistema de la IA utilizaba los componentes que guardan el estado actual de la inteligencia artificial de las entidades, y mediante un bucle comprobaba uno a uno cada uno de los elementos de la enumeración de estados declarada.

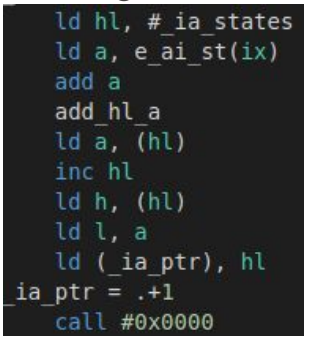
Figura 1



```
;e_ai_st_noAI        = 0
;e_ai_st_stand_by    = 1
;e_ai_st_move_to     = 2
;e_ai_st_patrol      = 3
;e_ai_st_seek        = 4

_ia_states::
    .dw nullptr
    .dw entity_ai_sys_stand_by
    .dw entity_ai_sys_move_to
    .dw entity_ai_sys_patrol
    .dw entity_ai_sys_seek
```

Este sistema no era óptimo dado que haciamos muchas comprobaciones innecesarias para determinar cuál era la acción inmediata que debía realizar la IA en cada momento.

Figura 2



```
    ld hl, #_ia_states
    ld a, e_ai_st(ix)
    add a
    add_hl_a
    ld a, (hl)
    inc hl
    ld h, (hl)
    ld l, a
    ld (_ia_ptr), hl
_ia_ptr = .+1
    call #0x0000
```

Para su optimización hemos diseñado un array de punteros a función con las etiquetas de los métodos, y hemos utilizado el valor numérico del estado de la IA como desplazamiento para acceder directamente al método requerido, sin realizar comprobaciones de cada uno de dichos estados.

Sensor de proximidad en la IA del Alien

Para la implementación de un sensor de proximidad para la entidad Alien, hemos modificado parte del código implementado para el sistema de colisiones.

Simplemente hemos ampliado en 10 unidades los cálculos sobre las componentes coordenadas para luego actualizar las componentes objetivo y el estado de la IA del Alien.

Figura 1

```
entity_ai_sys_seek::
_ent_array_ptr_seek = .+2
    ld iy, #0x0000
    call entity_ai_sys_check_proximity
    ret c

    ld a, e_x(iy)
    ld e_ai_aim_x(ix), a
    ld a, e_y(iy)
    ld e_ai_aim_y(ix), a
    ld e_ai_pre_st(ix), #e_ai_st_seek
    ld e_ai_st(ix), #e_ai_st_move_to
    ret
```

Figura 2

```
entity_ai_sys_check_proximity::
    ld a, e_endx(iy)
    add #10
    sub e_x(ix)
    ret c

    ld a, e_endx(ix)
    add #10
    sub e_x(iy)
    ret c

    ld a, e_endy(iy)
    add #10
    sub e_y(ix)
    ret c

    ld a, e_endy(ix)
    add #10
    sub e_y(iy)
    ret
```

Así se hizo

Como se diseño la idea original

La idea original surgió a partir del popular "meme" que ocurrió en Septiembre de 2019, el plan de asaltar el Area 51 con "Naruto runners" (corriendo como Naruto). El diseño del videojuego es pues un pequeño asalto al Area 51 con el objetivo de rescatar a un Alien.

Tecnologías utilizadas

- **VSCode** - Programación del juego en ASM
- **GitHub** - Control de versiones
- **Winape y RetroVirtualMachine** - Emuladores Amstrad CPC
- **CDT2WAV** - Conversor cdt a wav usado para cargar el juego en un Amstrad CPC original
- **Wine** - Para poder ejecutar winape desde linux
- **Gimp** - Creación de sprites
- **Tiled** - Generación de tilemaps
- **ArcosTracker 1.0** - Generación de música

Problemas encontrados y soluciones adoptadas

Conservación del fondo en los mapas. Para poder conservar el fondo tras el pintado y borrado de los sprites se modificó el sistema de render para que, en lugar de pintar los sprites con la función de cpctelera *draw_Sprite* con su posterior borrado pintando una caja con *draw_SolidBox*, pintara los sprites con la función *draw_SpriteBlended* en concreto con la opción de XOR para que aplicara la función XOR pixel a pixel del sprite con su respectiva posición en memoria de vídeo para poder borrarlo en cualquier momento aplicando la misma función en el mismo punto conservando el fondo.

Parpadeos en los sprites (flickering). Otro de los problemas venía porque había sprites que no se pintaban y otros que parpadeaban ya que al querer pintar diversas entidades el render tardaba más en pintar que el render en pasar.
Lo solución a este problema fué incluir un doble buffer para ir intercambiando de donde va a ir leyendo los datos el raster cuando quiera pintar la pantalla, consiguiendo así que siempre se esté pintando un mapa con todos los sprites pintados sobre el.
Para que funcionara correctamente también se tuvo que añadir un puntero extra en cada entidad para tener el puntero actual de la entidad en el buffer visible y otro en la posición en la que está pintada en el buffer trasero consiguiendo así que sepamos la posición de borrado para el buffer trasero.

Fallos en la reproducción de música. Al incluir la música, tanto al cargar los niveles como salir de los menús la música se desfasaba.
Para solventar este fallo se utilizaron las interrupciones de forma que se puede conseguir una constancia a la hora de reproducir la misma.

Tiempo de desarrollo del proyecto y sus fases

Horas invertidas

- Juan Carlos López Gutiérrez
  - 15 horas - aprendizaje de código máquina
  - 30 horas - clases presenciales
  - 20 horas - revisión de sesiones
  - 80 horas - trabajo individual

- Antonio Aguirre Ivorra
  - 10 horas - Videos de aprendizaje
  - 20 horas - Clases presenciales
  - 20 horas - Revisión sesiones RA y V1
  - 70 horas - Trabajo individual

- Jorge Amil Pérez Ponsoda
  - 10 horas - Videos de aprendizaje
  - 25 horas - Clases presenciales
  - 25 horas - Revisión sesiones
  - 100 horas - Trabajo Individual

Distribución en el tiempo

- Juan Carlos López Gutiérrez
  - Sesión 1 - Aprendizaje en paralelo al proyecto.
  - Sesión 2 - Aprendizaje en paralelo al proyecto.
  - Sesión 3 - Creación de menús básicos, transición para empezar a jugar y pequeña mejora del input.
  - Sesión 4 - Mejora del dibujado y borrado de los menús con estructuras para guardar información de cada uno y agregación de vida a las entidades.
  - Sesión 5 - Agregación de comprobación de game over con la vida del player mostrando el menú de derrota. Refactorización del bucle principal del juego para adaptarlo al progreso de diferentes niveles.
    Update de render conservando el fondo pintando y borrando entidades (blended XOR).
    Borrado de pantalla completo, creación de otro nivel, corrección de errores, conservación de la vida del jugador entre niveles y agregación de array de punteros en render.
  - Sesión 6 - Corrección de errores, intento de pintado de mapa con un tile (cancelado por otra propuesta mejor), creación de entidad "puerta" para avanzar de nivel.
    Creación de doble buffer con agregación de puntero a la posición en el buffer secundario de las entidades y adaptación del sistema de render.
  - Sesión 7 - Creación de manuales de usuario, ayuda al cambio de carga de fondo para cada nivel y memoria.
    Control de interrupciones e inclusión de controles por joystick.

- Antonio Aguirre Ivorra
  - Sesión 1 - Aprendizaje previo al proyecto.
  - Sesión 2 - Aprendizaje previo al proyecto junto al diseño de sprites iniciales.
  - Sesión 3 - Finalización del aprendizaje, ayuda en los diversos apartados en desarrollo.

- Sesión 4 - Producción de sistemas de colisiones y proyectiles inicial, que serían depurados y corregidos más tarde.
- Sesión 5 - Coproducción del sistema de colisiones final, coproducción del sistema de proyectiles final. Corrección general de errores.
- Sesión 6 - Implementación del sistema de mapas por tiles, corrección y mantenimiento del código anteriormente realizado.
- Sesión 7 - Diseño e implementación de niveles mediante mapas por tiles, producción de la música final.

- Jorge Amil Pérez Ponsoda - aprox 25 horas / semana
  - Sesión 1 - Aprendizaje, diseño de la idea original inicial y producción de ES inicial.
  - Sesión 2 - Aprendizaje, cambio del diseño de la idea original inicial y producción de sistemas de render y de físicas iniciales.
  - Sesión 3 - Aprendizaje y producción de sistema de inteligencia artificial inicial, mantenimiento del repositorio, propuesta de MVP.
  - Sesión 4 - Cambios estructura de datos y producción de sistema de patrullas, depuración y correcciones de código, mantenimiento del repositorio, propuesta de MVP.
  - Sesión 5 - Cambios estructura de datos, coproducción del sistema de colisiones, depuración y correcciones de código, mantenimiento del repositorio, propuesta de MVP.
  - Sesión 6 - Mejoras del sistema de inteligencia artificial y coproducción del sistema de proyectiles, depuración y correcciones de código, mantenimiento del repositorio, propuesta de MVP.
  - Sesión 7 - Documentación, mejoras del sistema de inteligencia artificial, mantenimiento del repositorio, propuesta de MVP.

Lecciones aprendidas

Costes sobre/infra-estimados

Aprendizaje de la tecnología. No estar familiarizados con el lenguaje ensamblador en Z80 ha supuesto para el grupo un coste temporal alto del tiempo requerido para ser dedicado a otras tareas del proyecto. La fase de aprendizaje acabó abarcando parte de la fase de preproducción. Sin embargo, su valor a corto plazo ha resultado incalculable, ya que ha facilitado las tareas de desarrollo y depuración, acortando los tiempos requeridos en dichos procesos del software.

Aciertos/fallos de planificación

Compromiso con el MVP. Concentrar nuestro esfuerzo en la obtención de un producto mínimo viable cada semana ha sido todo un acierto, a pesar de que ninguna de ellas hayamos conseguido cumplir con las tareas con las que nos comprometimos en las reuniones de planificación. Al finalizar la fase de producción, la suma de todos esos esfuerzos en favor de obtener un MVP han dado lugar a un producto de calidad, fiable y acabado.

También es importante destacar dentro del aspecto de planificación la comunicación entre los integrantes del grupo con el fin de planificar las distintas implementaciones de los sistemas de modo que facilite la complementación entre estos.

Toma de decisiones

Recortar expectativas iniciales. Estas tres palabras resumen perfectamente la motivación a la hora de tener que tomar decisiones en este proyecto. De la idea original hemos acabado recortando muchas funcionalidades como por ejemplo:

- Que nuestro jugador tuviera la capacidad de coger objetos y usarlos
- Que nuestro jugador tuviera habilidades especiales
- Que los enemigos tuvieran comportamientos especiales dependiendo de las acciones del jugador

- Que se pudiera interactuar con el objetivo a rescatar del juego
- Que el juego tuviera distintos finales posibles y que desbloquearan distintos sprites para el jugador