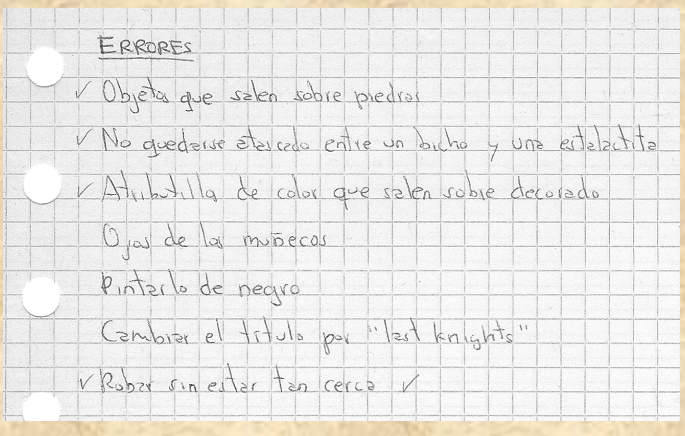
Vega Solaris. El desarrollo.

Fernando Sáenz Pérez. Noviembre 2013

Antecedentes

Aunque ya recogido en otros lugares, comenzaré resumiendo el contexto previo al desarrollo del juego. Con una experiencia básica en programación obtenida en sistemas como la calculadora *Casio FX-802P*, el cartucho de programación en ensamblador para *Philips Videopac G7000* y mi primer ordenador casero *Sinclair ZX81*, entra en juego la revolución del *Sinclair ZX Spectrum*. Aun siendo un pequeño ordenador personal, podríamos considerarlo como la videoconsola de más impacto de su época, ya que su uso fue dedicado mayoritariamente a los videojuegos. En mi caso, quizás al 50% (gracias a mi hermano Javier, que jugó un papel fundamental como *beta tester* en el desarrollo de *Vega Solaris*) compaginado con la programación.



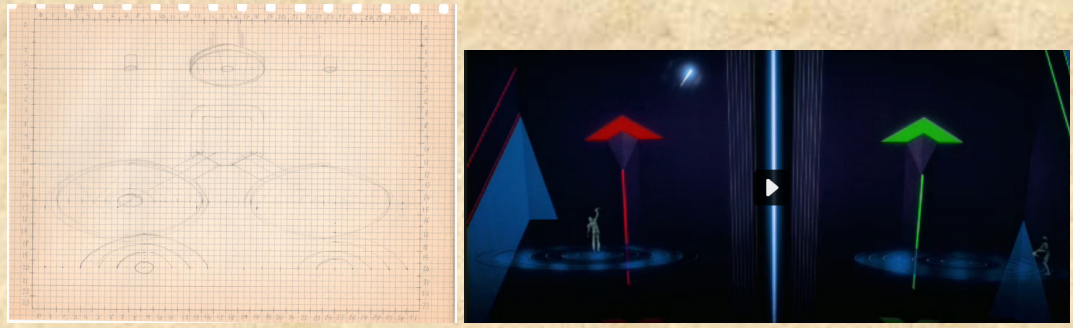
Algunos resultados del beta tester

Como muchos otros, nuestra situación era “de prestado”, o bien ocupaba un escritorio del cuarto de mi hermana o usaba el mueble de la máquina de coser de mi madre. Aquí colocaba la televisión analógica conectada al *Spectrum*, y este al radiocasete lector y grabador de cintas de programa, y también a un pequeño amplificador que construí para una mejor experiencia de sonido. Asimismo hice algún otro pinito en electrónica como la construcción de un modem que probé con éxito al enviar un programa a un amigo a través del teléfono (eso sí, haciendo mucho ruido porque usaba el micro y el altavoz para el acoplamiento).

¿Por qué nace Vega Solaris?

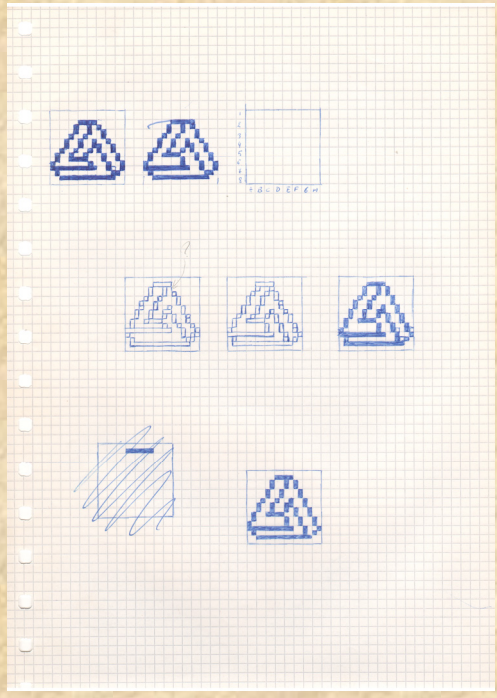
Me fascinaba la capacidad de algunas calculadoras y de los ordenadores de comportarse tal como uno los programase. Después de escribir innumerables programas en Basic y de afrontar el código máquina y el ensamblador, este se revelaba como el paso inmediato para conseguir movimientos fluidos en los juegos. Aunque existían los compiladores de Basic, eran bastante limitados y no suponían una alternativa real al desarrollo a bajo nivel. Aun así, escribí el programa Basic de las motos de Tron con aceleración y efectos de sonido que quedaron decentes al compilarlo. Recuerdo que al llevarlo a una tienda de Embajadores para su posible comercialización me preguntaron por qué lo había escrito en inglés (bueno, la mayoría de juegos estaban en ese idioma y pensé que tendría más ventas). Aunque no entendida en su época, la película Tron fue revolucionaria y me supuso una gran influencia. No en vano, también comencé el

desarrollo en código máquina del juego del frontón, con el mismo escenario que aparece en la película.



Diseño del juego de frontón basado en la película Tron

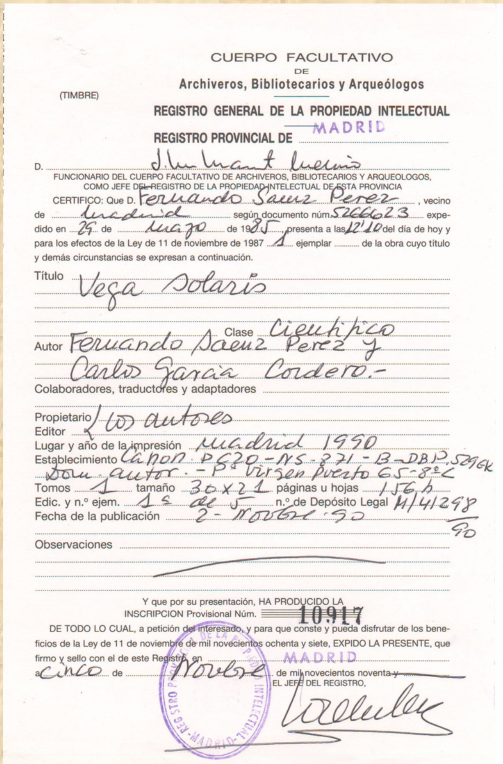
Por aquellos entonces cursaba primero de Ciencias Físicas en la universidad Complutense, donde conocí a un amigo cuyo hermano, Carlos, también estaba interesado en los videojuegos. Él sería el coautor de *Vega Solaris*, proporcionando gran parte de la idea, los gráficos, la carga del juego y del mapa, y el minijuego de marcianitos. También comenzó los gráficos del juego de frontón, aunque no lo terminamos. Otro amigo de la facultad realizó un gran trabajo en el diseño de las figuras estilo Escher de los cristales.



Estudios de uno de los cristales

En ese momento ya había desarrollado una rutina de "sprites", un elemento indispensable en los juegos para controlar gráficos en movimiento. Con un sprite se permite definir un objeto de ciertas dimensiones y combinarlo con un fondo y otros sprites siguiendo un sistema de prioridades que determina qué objeto oculta a otro por la capa a la que pertenece. Algunos ordenadores contaban con estas rutinas en ROM e incluso disponían de hardware especializado para su tratamiento, pero no era el caso del *Spectrum*. Escribí un artículo para la revista *ZX* describiendo esta rutina en ensamblador (recuerdo entrar en las oficinas y encontrarme al entonces joven director de la revista, Simeón Cruz, jugando y comentando un juego). Sin embargo, no se llegó a publicar

porque en otro artículo que sí se publicó (*Quinielas*) no pagaron lo esperado y un amigo abogado (demasiado escrupuloso en su trabajo) les envió un escrito que prácticamente imposibilitaba su publicación. Aun conservo este artículo y otro, *Tierra*, para la representación de los continentes en una esfera. Todos estos programas y artículos, además de *Vega Solaris*, están incluidos en el registro de la propiedad intelectual como obras de carácter científico. En aquel momento casi no se sabía qué era el software.



CUERPO FACULTATIVO
DE
Archiveros, Bibliotecarios y Arqueólogos

(TIMBRE)

REGISTRO GENERAL DE LA PROPIEDAD INTELECTUAL

REGISTRO PROVINCIAL DE MADRID

D. ...

FUNCIONARIO DEL CUERPO FACULTATIVO DE ARCHIVEROS, BIBLIOTECARIOS Y ARQUEOLOGOS, COMO JEFE DEL REGISTRO DE LA PROPIEDAD INTELECTUAL DE ESTA PROVINCIA

CERTIFICO: Que D. Fernando Saenz Perez, vecino de Madrid según documento núm. 5266623 expedido en 29 de Mayo de 1985, presenta a las 12'10 del día de hoy y para los efectos de la Ley de 11 de noviembre de 1987 1 ejemplar ...... de la obra cuyo título y demás circunstancias se expresan a continuación.

Título Vega Solaris

Clase Cientifico

Autor Fernando Saenz Perez y Carlos Garcia Cordero.-

Colaboradores, traductores y adaptadores

Propietario / Editor Los autores

Lugar y año de la impresión Madrid 1990

Establecimiento Canon PC20-NS-871-B-DBP.529GK
Dom. autor. - Pº Virgen Puerto 65-8ºC

Tomos 1 tamaño 30x21 páginas u hojas 156 h

Edic. y n.º ejem. 1ª de 5 n.º de Depósito Legal M/41298/90

Fecha de la publicación 2-Novbre-90

Observaciones

Y que por su presentación, HA PRODUCIDO LA INSCRIPCION Provisional Núm. 10917

DE TODO LO CUAL, a petición del interesado, y para que conste y pueda disfrutar de los beneficios de la Ley de 11 de noviembre de mil novecientos ochenta y siete, EXPIDO LA PRESENTE, que firmo y sello con el de este Registro, en MADRID a cinco de Novbre de mil novecientos noventa y ......

EL JEFE DEL REGISTRO,

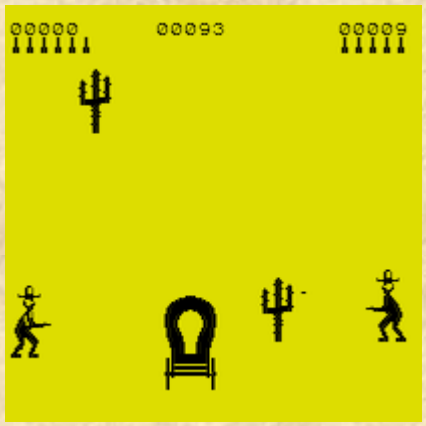
REGISTRO PROVINCIAL DE LA PROPIEDAD INTELECTUAL -MADRID-

Registro de la propiedad intelectual de Vega Solaris

Mi hermano y yo éramos ávidos consumidores de videojuegos (pocos originales porque eran muy caros para la economía doméstica y por otro lado fáciles de copiar de cinta a cinta) y en particular de los multijugadores como *Match Day, Spy vs. Spy* y *High Noon,* porque no es lo mismo ganar a una máquina que a tu propio hermano (sobre todo haciendo trampas al pulsar simultáneamente varias teclas, lo que interfería en la lectura de teclado). Por otro lado, *Ultimate*, con su serie de juegos insuperables como *Atic Atac* y *Sabre Wulf*, fueron una gran motivación para desarrollar *Vega Solaris*, que reúne los elementos de ambos tipos de juegos. La idea de su desarrollo era, por tanto, inevitable.

Sabre Wulf

Atic Atac



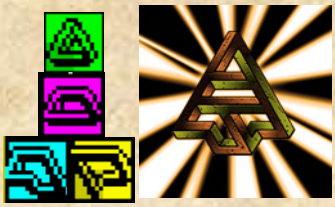
High Noon

Spy vs. Spy

Match Day

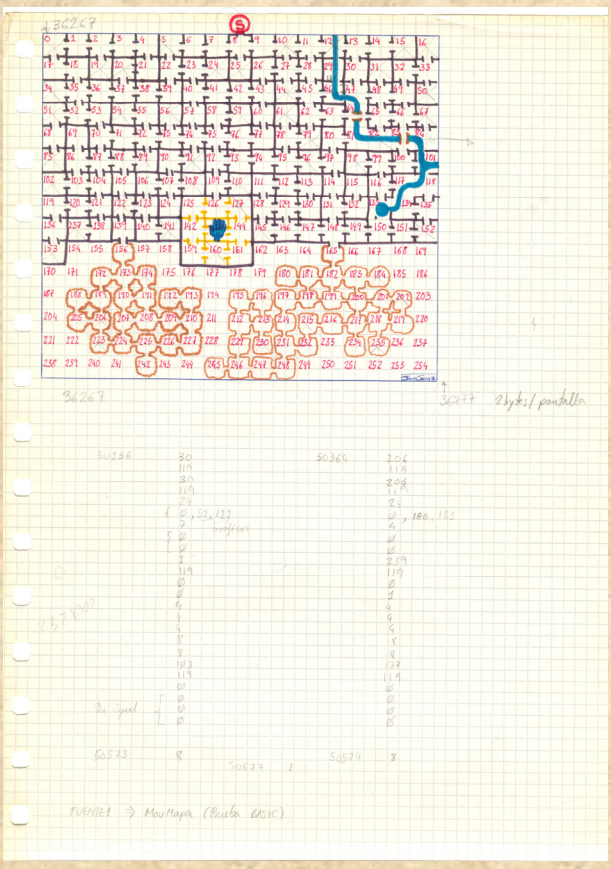
El juego

Con estos antecedentes se fragua *Vega Solaris*: un juego para dos jugadores con estética y trasfondo *Ultimate*, cuyo objetivo es conseguir conformar el talismán de *Vega Solaris* a partir de los cuatro cristales en que está fragmentado (tanto estos fragmentos como el talismán, en forma de punta de flecha, eran figuras "imposibles" estilo Escher o Penrose) y llevarlos a la pantalla inicial en un tiempo limitado.



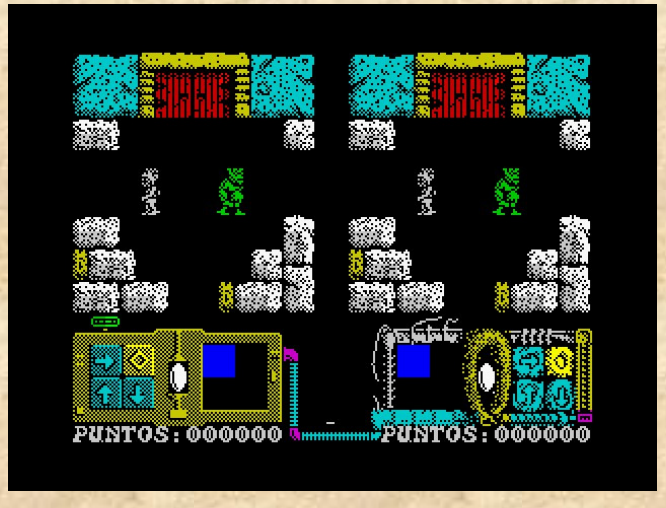
Los cristales y el talismán

Estos cristales están repartidos aleatoriamente en un mundo con varios escenarios, que incluyen una selva, cuevas y un templo, y que se reparten entre doscientas once pantallas. Dadas las limitaciones de memoria, Carlos sugirió recortar el mapa. Sin embargo, preferí exprimir el código para disponer del mayor mapa posible. El ganador del juego es el personaje (humano o extraterrestre) que haya conseguido llegar a esta pantalla inicial con los cuatro cristales en los bolsillos antes de un tiempo límite muy, muy ajustado, como correspondía a la dificultad de los juegos de aquella época.



Boceto del mapa

Hay dos personajes en la búsqueda de este talismán: un humano y un extraterrestre a los que se les debería reconocer por su aspecto, que incluyó distintos diseños hasta su versión definitiva. Este es precisamente uno de los alicientes del juego, ya que se puede jugar contra otro jugador o contra el *Spectrum* (o incluso dejar el juego completamente en sus manos), para lo cual hube de desarrollar una rutina de inteligencia artificial que permitiese al personaje moverse libremente, recoger y dejar objetos, luchar y robar al oponente. Como se ve en la siguiente figura, la pantalla está dividida en dos paneles que corresponden a cada personaje; el panel izquierdo es el del humano y el derecho, el del extraterrestre. Esta figura muestra la pantalla inicial de la que parten ambos.



Pantalla de inicio

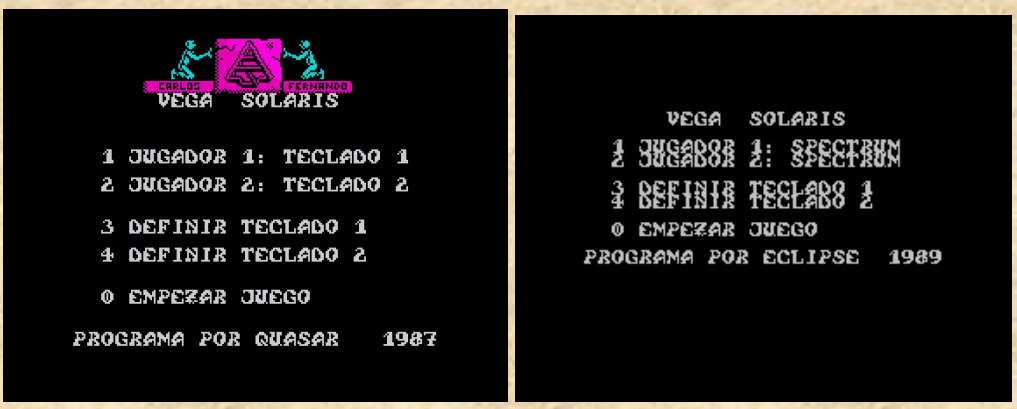
En la parte inferior se encuentra información del contenido de los únicos cuatro bolsillos (inventario) de los personajes en que pueden guardar y llevar los objetos que encuentren a su paso.

Durante el recorrido encontrarán no sólo estos cristales, sino también armas con distinto nivel de daño que pueden usar contra el otro personaje y contra los bichos que aparecen cada vez con mayor frecuencia. Tanto los personajes como los bichos pueden infligir daños que se representan con colores en el indicador de energía del personaje. Estos colores varían de blanco brillante (sin daños) a negro (consumida toda la energía). Cuando se alcanza el negro, el personaje debe reponerse, quedando inconsciente durante algún tiempo, en el cual el otro personaje puede robarle los objetos que posea.

Armas, escudos, conjuros y comida

Bichos. El alacrán era especial

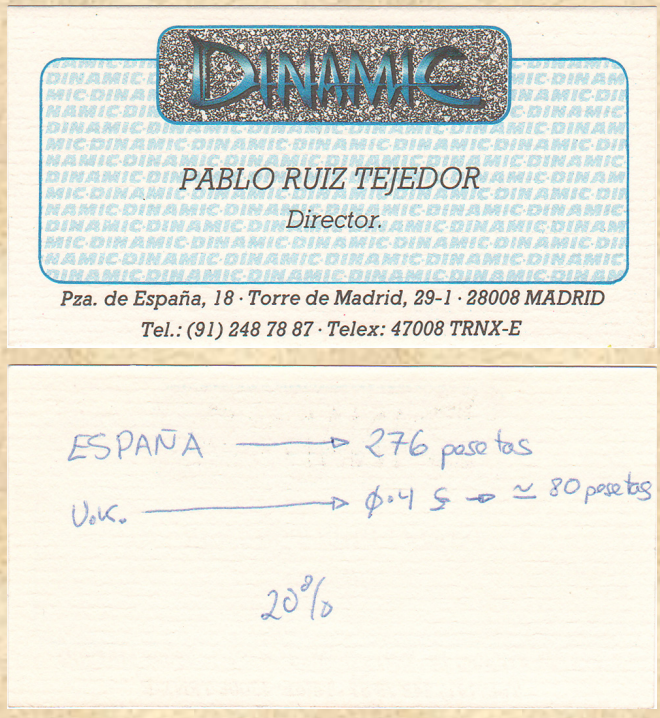
En el menú principal del juego se puede seleccionar el personaje que se desee manejar y si el oponente será otro jugador o el *Spectrum*. También el tipo de control: tanto teclado como *joystick*. En la versión definitiva el menú aparece comprimido y sin el gráfico de la versión beta:



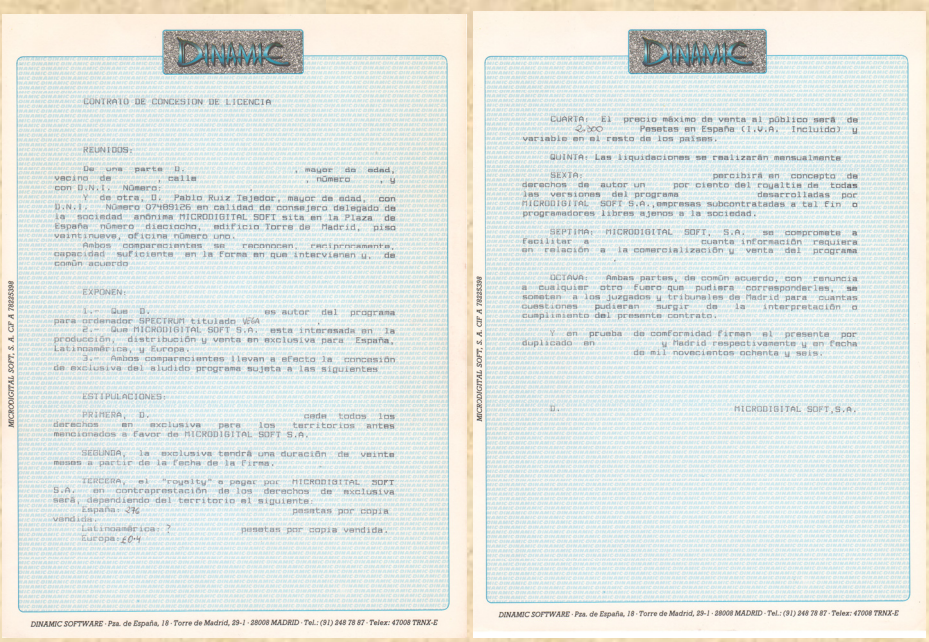
Menú principal en la versión beta (izquierda) y la definitiva (derecha)

Como se puede observar, cada uno de los caracteres del conjunto que diseñó Carlos para el juego ocupaba las ocho líneas de una fila, por lo que al situar unas líneas a continuación de otras no se leen bien las entradas del menú. Sin embargo, esto se tuvo que hacer por problemas de memoria. En la versión definitiva se almacena el panel de estado dibujado a continuación (debajo) del menú, pero con los atributos de color para la tinta y el fondo a negro. Esto posibilitaba guardar en otra ubicación de memoria (distinta de la de vídeo) tan sólo estos atributos de color en lugar de la imagen completa, lo cual ahorraba unos valiosos bytes.

Otra diferencia es el nombre del equipo de desarrollo: inicialmente nos llamamos Quasar y después cambiamos a Eclipse. Nunca incluimos el nombre de Dinamic ni de otra distribuidora al no publicarse. Aun conservo la demo que llevamos a Dinamic en una cinta reutilizada del INBAD, la tarjeta que nos dio Pablo con las condiciones de venta por royalty y el modelo de contrato. Si bien yo estuve en las oficinas de la Torre de Madrid dos o tres veces, Carlos fue alguna vez más.



Tarjeta de Pablo Ruiz con las condiciones



CONTRATO DE CONCESION DE LICENCIA

REUNIDOS:

De una parte D. , mayor de edad, vecino de , calle , número , y con D.N.I. Número .

Y de otra, D. Pablo Ruiz Tejedor, mayor de edad, con D.N.I. Número 07489126 en calidad de consejero delegado de la sociedad anónima MICRODIGITAL SOFT sita en la Plaza de España número dieciocho, edificio Torre de Madrid, piso veintinueve, oficina número uno.

Ambos comparecientes se reconocen recíprocamente, capacidad suficiente en la forma en que intervienen y, de común acuerdo

EXPONEN:

1.- Que D. es autor del programa para ordenador SPECTRUM titulado VEGA .

2.- Que MICRODIGITAL SOFT S.A. esta interesada en la producción, distribución y venta en exclusiva para España, Latinoamérica, y Europa.

3.- Ambos comparecientes llevan a efecto la concesión de exclusiva del aludido programa sujeta a las siguientes

ESTIPULACIONES:

PRIMERA. D. cede todos los derechos en exclusiva para los territorios antes mencionados a favor de MICRODIGITAL SOFT S.A.

SEGUNDA. la exclusiva tendrá una duración de veinte meses a partir de la fecha de la firma.

TERCERA. el "royalty" a pagar por MICRODIGITAL SOFT S.A. en contraprestación de los derechos de exclusiva será, dependiendo del territorio el siguiente:

España: 2% pesetas por copia vendida.
Latinoamérica: ? pesetas por copia vendida.
Europa: £0.4

MICRODIGITAL SOFT, S. A. CIF A 78283398

DINAMIC SOFTWARE · Pza. de España, 18 · Torre de Madrid, 29-1 · 28008 MADRID · Tel.: (91) 248 78 87 · Telex: 47008 TRNX-E

CUARTA: El precio máximo de venta al público será de pesetas en España (I.V.A. incluido) y variable en el resto de los países.

QUINTA: Las liquidaciones se realizarán mensualmente

SEXTA: percibirá en concepto de derechos de autor un por ciento del royalty de todas las versiones del programa desarrolladas por MICRODIGITAL SOFT S.A.,empresas subcontratadas a tal fin o programadores libres ajenos a la sociedad.

SEPTIMA: MICRODIGITAL SOFT, S.A. se compromete a facilitar a cuanta información requiera en relación a la comercialización y venta del programa

OCTAVA: Ambas partes, de común acuerdo, con renuncia a cualquier otro fuero que pudiera corresponderles, se someten a los juzgados y tribunales de Madrid para cuantas cuestiones pudieran surgir de la interpretación o cumplimiento del presente contrato.

Y en prueba de conformidad firman el presente por duplicado en y Madrid respectivamente y en fecha de mil novecientos ochenta y seis.

D. MICRODIGITAL SOFT,S.A.

MICRODIGITAL SOFT, S. A. CIF A 78283398

DINAMIC SOFTWARE · Pza. de España, 18 · Torre de Madrid, 29-1 · 28008 MADRID · Tel.: (91) 248 78 87 · Telex: 47008 TRNX-E

Modelo de contrato

Carlos (en primer término) trabajando en Dinamic

La versión beta apareció en algún momento en Internet de manos de no sabemos quién. Probablemente de alguien a quien dejásemos probar el juego. En las presentaciones a las distintas empresas a las que nos dirigimos para comercializarlo no dejamos copia; incluso apagábamos el ordenador tras la demostración. Creo que fue cuando fuimos a Made in Spain a casa de César Menéndez (aun tengo anotado su teléfono en los papeles antiguos y responsables del magnífico *Sir Fred*), en la ciudad de los periodistas, cuando vimos casi imposible realizar las conversiones que nos pedían para poder comercializarlo.

Para la carga del juego usamos el procedimiento habitual: escribir LOAD “”, insertar la cinta y pulsar PLAY en el casete. Para darle un poco de distinción, Carlos desarrolló en la fase final una rutina especial de carga que dibuja la pantalla por líneas consecutivas, en lugar del proceso de dibujo habitual por tercios de pantalla y por líneas de filas. Además, aburridos de esperar la carga como en cualquier otro juego, una vez dibujada la pantalla de carga, aparece una versión de *Space Invaders* para jugar mientras dure el resto del proceso de carga. El objetivo de este minijuego es simplemente obtener el máximo número de puntos matando marcianitos. La puntuación indica el número de ellos abatidos y se hace cero si alguno consigue llegar a nuestra posición. Los patrones de movimiento cambiaban ligeramente según aparecían nuevas hordas de marcianos.

Minijuego durante la carga de cinta

Fuentes documentales

Como para muchos otros programadores, *The Complete Spectrum ROM Disassembly* (*Melbourne House*, 1983) era de obligada referencia no sólo para aprender sino para evitar reescribir cosas que ya estuviesen en la ROM y ahorrar memoria. Con un libro inglés sobre el ensamblador del *Z80* aprendí en el *ZX81* las instrucciones para programar en código máquina. Los artículos de las revistas *ZX* (la serie "Aprendiendo el código máquina" y otros específicos) y de *Microhobby* eran de gran ayuda (los artículos de revisiones de juegos en *Micro Manía* me resultaban realmente inspiradores). Una vez conocidos el lenguaje Basic (gracias a los manuales del *ZX81* y del *Spectrum*) y cómo implementar algoritmos, sería fácil programar en código máquina. Sin embargo, esto traía otras complicaciones prácticas.

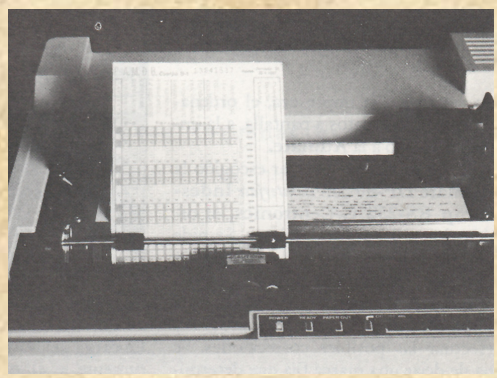
Algunas fuentes documentales

Las herramientas

En cuanto a hardware, contaba obviamente con el *Spectrum*, pero con el teclado dk'tronics de teclas mecánicas que reemplazó al antiguo de "goma" cuando empezó a fallar (notablemente las teclas Q, A, O, P y N conocidas por todos los jugadores). También disponía de un televisor en color de 14 pulgadas de Philips que reemplazó a su vez a otro de iguales dimensiones pero en blanco y negro, y que sería de gran ayuda para ver el efecto del color en los gráficos. Carlos, aun sin disponer de televisor en color, desarrolló un magnífico trabajo gráfico. Sobre la mitad del desarrollo, mi padre me compró un microdrive que resultó de gran ayuda porque cargaba rápidamente bloques de memoria, fundamental después de experimentar un cuelgue y recuperar el estado anterior. También disponía de una impresora de agujas, la Admate DP-80, con la que imprimimos los gráficos y el código para el registro de la propiedad (también se usó para rellenar los boletos en el programa Quinielas).

Teclado dk'tronics. Las pegatinas se despegaban de tanto jugar

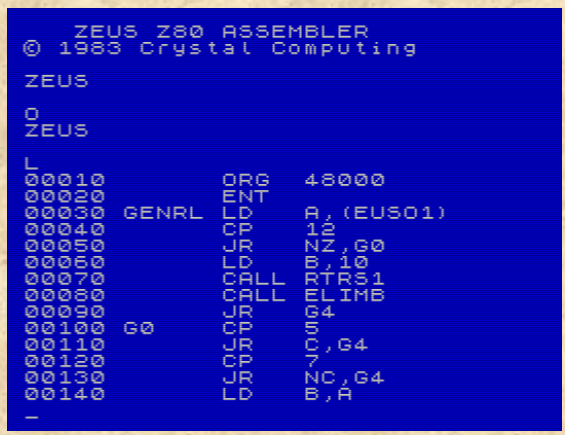
Televisor Philips B/N



Impresora Admate DP-80 rellenando boletos de quinielas

El entorno de trabajo de Fernando con el televisor en color

En cuanto a software, la elección común era la pareja GENS y MONS, ensamblador y desensamblador respectivamente (David Link, 1983) que tenían la gran ventaja de ser reubicables; es decir, no contenían ninguna referencia absoluta a memoria, sino que todas ellas eran relativas. Esto simplificaba el desarrollo al encontrar un hueco para ubicarlo. Sin embargo, después de ver la película *Juegos de guerra,* me quedé con la imagen (entonces inexistente en el panorama informático doméstico) de los editores libres, que permitían escribir en cualquier lugar de la pantalla, al igual que ZEUS (Crystal Computing, 1983), el sistema por el que me decantaría. Además permitía escribir completa o abreviadamente los comandos, las instrucciones y comandos se escribían carácter a carácter (a diferencia de Basic, cuyas palabras reservadas se escribían de golpe con una combinación de teclas), y revolucionaba los colores con letra blanca sobre fondo azul, una combinación que según estudios relajaba la vista durante las largas horas de programación (algo que realmente noté y agradecí). Al no ser reubicable, debía tener especial cuidado en el lugar en el que se almacenaba tanto el ensamblador como los fuentes que cargaba (algunos crecieron tanto como para invadir zonas de código máquina).

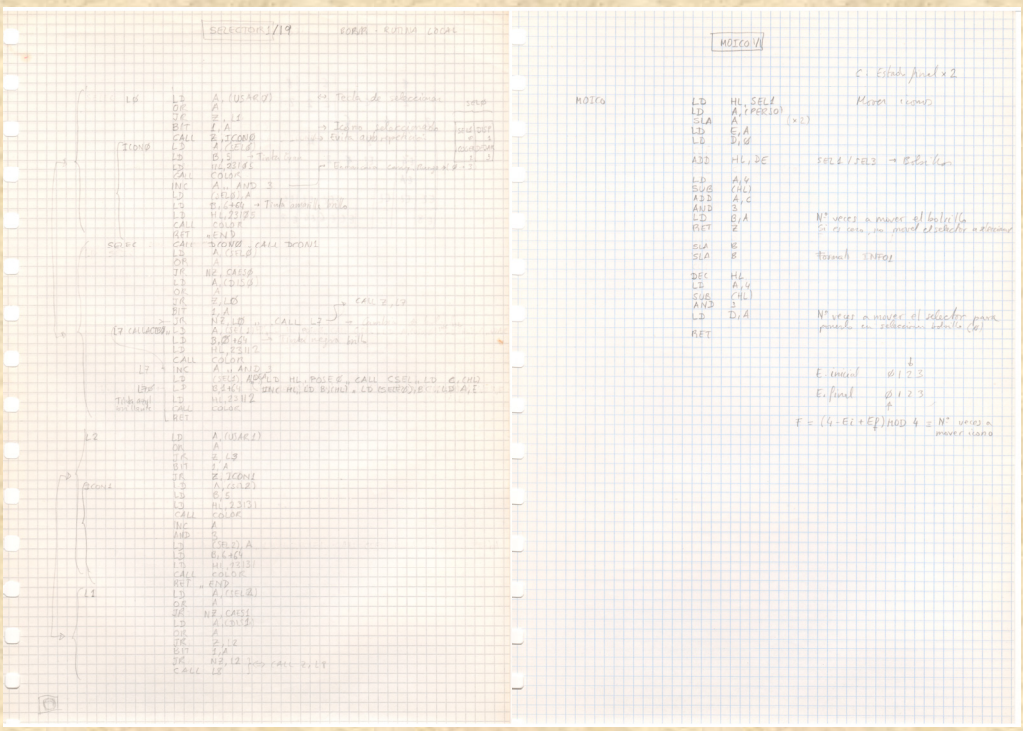


El ensamblador ZEUS y el bucle principal de juego de Vega Solaris

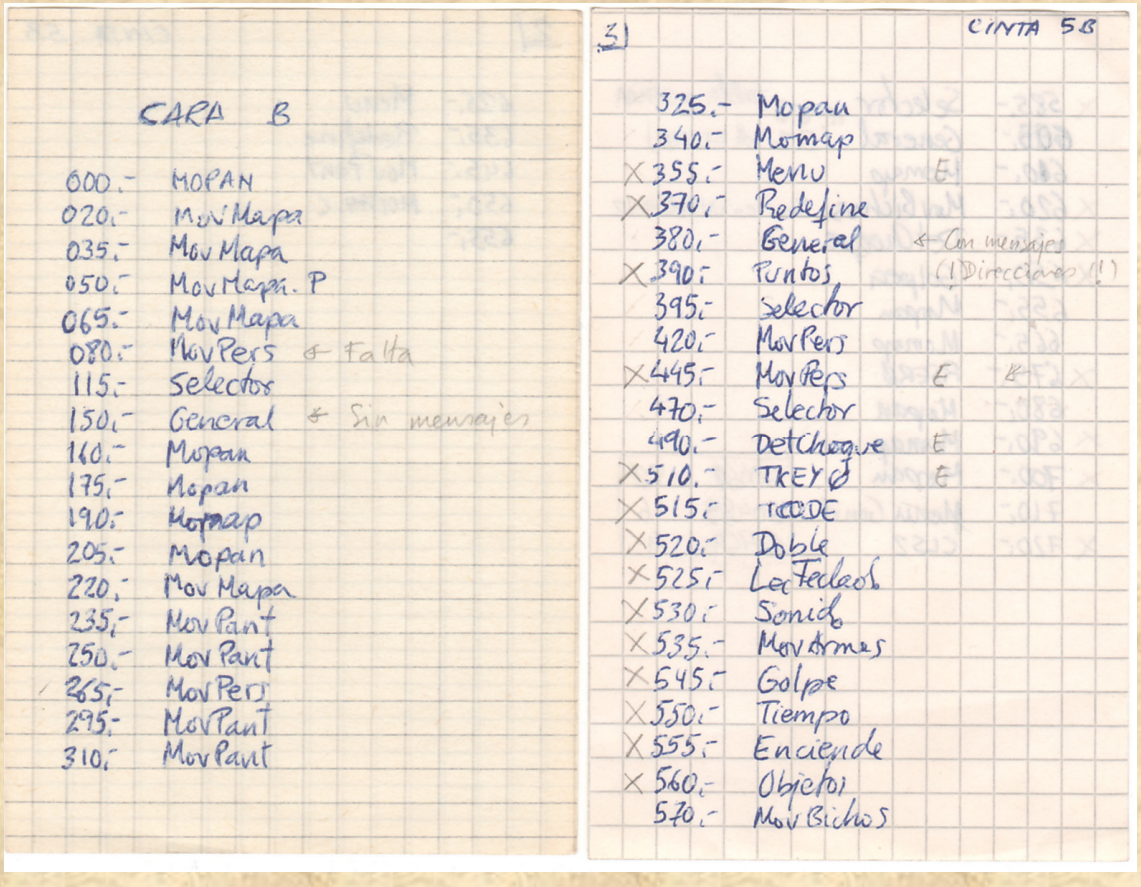
Además de todo esto disponía de lápiz, papel y mucha goma de borrar.

El desarrollo, o "divide y vencerás"

Para resolver una parte del juego, que dividía en las denominadas "rutinas", generalmente pensaba el mejor modo de resolverlo, tanto en tiempo como en memoria ocupada. Para ello me venía muy bien la descripción del repertorio de instrucciones del *Z80* en el que se indicaba el número de bytes y de ciclos necesarios para almacenar y ejecutar cada una de las instrucciones. Estos datos los consultaba con cada vez menos frecuencia porque se iban alojando en mi "memoria caché". Una vez aclarado cómo resolverlo, el siguiente paso era describirlo en papel y sacar a relucir los aspectos en los que no había pensado, como el uso eficiente de los registros para evitar usar variables en memoria (la goma tenía un papel fundamental en esta fase). Disponer del código en papel también aportaba una visión más general que en cualquier ensamblador, ya que la ventana de código que se podía listar se reducía a poco más de veinte líneas. Sólo cuando había repasado detalladamente la rutina (ejecutándola mentalmente y siguiendo la traza del estado de los registros y de las variables involucradas) pasaba a escribirla en el ordenador. Después de revisar la transcripción del código ensamblador, lo grababa en cinta sin sobrescribir versiones anteriores para evitar el temido "R Tape loading error" (conservo seis cintas de 90 grabadas en ambas caras con los fuentes, es decir, casi nueve horas).



Algunas rutinas quedaban mejor en papel que otras



Fuentes grabados en cinta con indicación del contador de cinta

La grabación del fuente ensamblador no era otra cosa que un volcado de memoria a cinta especificando una dirección de inicio y un tamaño, proporcionado por el ensamblador mediante el comando T. Este procedimiento se hacía necesario para mí en cuanto que un simple fallo ocasionaba una gran pérdida de tiempo. Hay que recordar

que el concepto actual de depuración de programas no existía en el nivel del ensamblador. No había herramientas para examinar al vuelo los registros de la CPU. Sólo se podía observar el estado final según quedase la memoria (siempre que el sistema no se colgase, claro). Así, este procedimiento me recuerda a cómo se usaban los antiguos ordenadores de tarjetas perforadas en la primera misión espacial a la Luna: el cálculo de una trayectoria duraba más de 8 horas y un error en el programa podía suponer no conseguir el objetivo de colocar a un hombre en la superficie lunar en la década de los 60. Antes de probar era necesario asegurarse de que no hubiese errores. No obstante, eran inevitables.

Nosotros no disponíamos de un sistema de ensamblador cruzado como sí tenían en Dinamic, por ejemplo, lo que les permitía desarrollar en una máquina, transferir el código ensamblado al *Spectrum* y probar. Si este se colgaba, se continuaba desde el sistema de desarrollo y se volvía a transferir. En nuestro caso esto suponía volver a cargar el ensamblador, los segmentos de memoria con gráficos y código necesarios para probar la rutina en concreto y su fuente en ensamblador. Todo ello desde cinta. Con el microdrive las cosas fueron un poco mejor, aunque para funcionar era necesario tener en memoria sus rutinas del sistema de archivos.

El desarrollo moderno de videojuegos nos enseña en particular que los juegos multijugador se deben diseñar como tales desde el principio. No se debe partir de un diseño de un solo jugador para extenderlo a multijugador porque implica demasiados cambios en partes cruciales. Afortunadamente la idea de nuestro juego estaba perfectamente perfilada como tal. Por otra parte, los desarrollos actuales habituales usan la concurrencia con los hilos, facilitando la programación y observando a los objetos como entidades con un comportamiento propio. En los 80 no se disponía de estos lujos y era necesario implementar explícitamente la gestión de la concurrencia con un bucle principal en el que se atendían secuencialmente las distintas tareas, a las que se dedicaba un cierto tiempo.

En el caso de *Vega Solaris*, este bucle tenía la etiqueta nemotécnica GENRL (de "general", siempre andaba buscando acrónimos que cupiesen en 5 caracteres, el máximo tamaño de los identificadores) y le correspondía la ubicación 63.906 de memoria, con una ocupación de 645 bytes (la longitud de palabra de memoria era de 1 byte, aunque se podía acceder con una única instrucción a dos posiciones consecutivas en dos ciclos de lectura). Otro bucle menos complicado era el menú de inicio en el que se escogían las opciones y se podía iniciar el juego: MENU:64.551:891 (lo que viene a indicar etiqueta:dirección de memoria:tamaño). Ocupa más que GENRL simplemente porque incluía otras rutinas (gestión de la redefinición del teclado, submenús, etc.). El bucle principal incluía la gestión del tiempo general, el movimiento del personaje en sí (con la ralentización del movimiento de cada personaje si le había picado el alacrán), de las armas, la gestión de los objetos y de los bichos, y la detección de fin de juego por tiempo o por éxito. Casi todo se gestionaba por rutinas externas. Realmente estas rutinas tenían un punto de entrada principal pero podían tener otros secundarios. Esto evitaba tener muchos más programas en ensamblador almacenados en cinta. En la compactación final había catalogados 36 de estos programas, en los que se incluían 20 referencias internas como subrutinas. Habitualmente las variables se definían en las propias rutinas a las que pertenecían por naturaleza, aunque también se referenciaban desde otras. Hay documentadas más de 80 variables globales.

Dentro de las rutinas principales que se llamaban desde el bucle principal se encuentran las del movimiento de los personajes (MOPE0:44.713:2.149), la gestión de objetos (OBJE0:56.021:603), el movimiento de las armas arrojadizas (MOAR0:50.963:871), la

gestión del inventario (SELCT:53.853:2.168) y el movimiento de los bichos (MOBI0:47.266:1.337). Las detecciones de colisiones (CRASH:50.587:178) para los objetos móviles las realizamos examinando la línea de píxeles a los que se desplazaría el objeto. Esto suponía el peligro de "comernos" el fondo al ser estático y poder avanzar en diagonal. Así que todos los gráficos y objetos inmóviles que podían estar en contacto con los objetos móviles tenían al menos una fila de píxeles contra la que se podía detectar la colisión. La gestión de las acciones a realizar cuando ocurrían las colisiones se llevaban a cabo por otras rutinas (para las armas, personajes y bichos: DCHA0:52.313:1.540). Por ejemplo, en esta rutina se trataban cosas del estilo de cómo actuar cuando se llevaba la espada, se lanzaba un puñetazo, se arrojaba un arma o un bicho te alcanzaba. La activación de una acción (como los puñetazos) la gestionaba la rutina de golpes (GOLP0:51906:150). La gestión del inventario incluía también una detección de proximidad en baja resolución según las coordenadas de los objetos para comprobar que se pudiesen coger. Asimismo, era necesario localizar huecos libres con una detección basada en atributos para determinar dónde dejar los objetos, comenzando a buscar en la ubicación más próxima. Los objetos que se recogían a lo largo del mapa (conjuros de invisibilidad y teletransporte, escudos y comida) se gestionaban con OBJE0:56.021:603. Se obtenían puntos (gestionado por PUNT0:48.656:135) al matar a los bichos o al dañar al oponente, y añadimos un cero al final para dar sensación de mayores logros, aunque nunca cambiase ni se escribiese esta última cifra.

Un arma que dio bastante trabajo fue la espada, puesto que nos obligó a usar sprites más grandes (SPRES:46862:208), nuevos gráficos y un movimiento adaptado a su uso (más lento). Escribimos una rutina de sprites específica para ella. Otra arma original que tuvimos que descartar por problemas de espacio en la versión final fue el hacha, que iba dando vueltas según se arrojaba. Al igual que otros conjuros, la flecha rebotaba con el fondo y desaparecía al contactar con cualquier otro objeto móvil, pero su gráfico debía adaptarse según su orientación en vuelo.

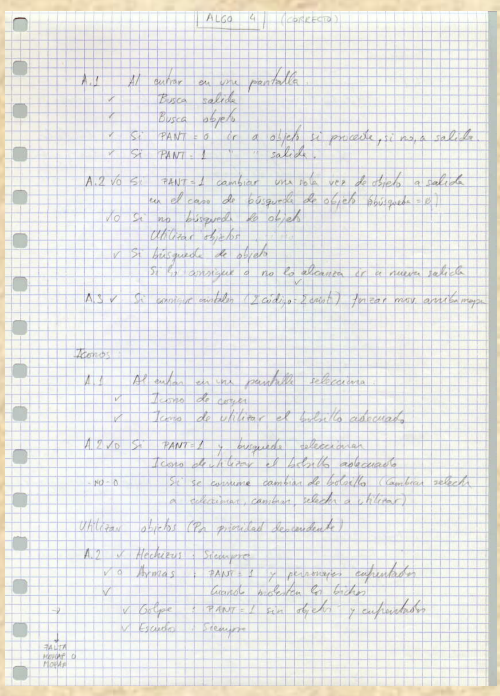
Aunque ya tenía preparada la rutina de sprites con el sistema de prioridades, al final no la usamos en favor de otra más simplificada, dado que no era necesario solapar distintos objetos móviles. Esto evitaba la necesidad de usar máscaras (para definir el color transparente) y consiguientemente ahorrábamos memoria. Por otra parte, esta rutina era lo suficientemente rápida como para evitar la colisión con el barrido de pantalla y así eliminar el efecto de gráficos partidos, incluso sin necesidad ni de sincronizarse con el impulso vertical PAL ni de usar la técnica del buffer de video (que consumía una gran cantidad de memoria).

Para implementar el conjuro de invisibilidad simplemente recurrimos a los atributos de color para dejar la pantalla en negro manteniendo los gráficos. La coincidencia de los personajes en la misma pantalla se manejaba con otra rutina (DOBLE:48.791:89) representando lo mismo en ambas vistas. En esta situación no aparecían los bichos puesto que era prácticamente imposible que cupiesen sin comprometer la jugabilidad. El escaso sonido que añadimos (pasos, golpes, apariciones de bichos, ...) se controlaba desde las rutinas de cada objeto llamando a una rutina genérica de sonido (SOUN0:48.603:53) que recibía varios parámetros (frecuencia, tiempo, envolvente y subida). Al no disponer de chip de sonido, había que programarlo todo (curioso que todo esto cupiese en tan solo 53 bytes). Como no éramos unos virtuosos compositores, ni conocíamos a ninguno, y tampoco andábamos sobrados de memoria, descartamos la música habitual que acompañaba a los juegos.

Los personajes respondían a las entradas externas proporcionadas por el teclado o el *joystick*. La rutina TECLS:49.408:78 permitía leer del teclado y de un tipo de *joystick*,

mientras que JOY0:36.655 leía de otro tipo. Además se permitían definir las teclas preferidas con REDEF:56.624:580. La lectura de entradas provenía de la rutina LECT0:50.765:198 para dispositivos cuando el personaje lo controlaba un jugador, o bien según decidía la inteligencia artificial (IA) si el personaje lo controlaba el *Spectrum*.

El algoritmo de IA fue un desarrollo personal muy edificante. Por su complejidad partí de una descripción de reglas en alto nivel usando pseudocódigo (lenguaje natural) que "compilé" posteriormente a ensamblador. El objetivo, dadas las limitaciones de memoria, fue crear un oponente lo suficientemente listo como para que te complicase el juego (aunque de hecho, era posible que ganase). Como detalles puedo indicar que no había espacio ni siquiera para conservar memoria del camino recorrido salvo para unas pocas pantallas y, ni mucho menos, para incorporar un algoritmo de búsqueda de caminos en un grafo. Me limité a implementar estrategias del tipo: "Dirigirse en dirección noroeste", "Intentar alcanzar la puerta; si no es posible, retroceder por un camino diferente", "Si un obstáculo impide tu paso, intentar sortearlo, si no es posible en un margen de tiempo razonable, renunciar", "Intentar coger un objeto de interés". Apliqué un sistema de prioridades para coger y usar los objetos, los de mayor prioridad para coger eran los cristales y, para usar, los que proporcionaban mayor poder de ataque y defensa. Estas reglas se implementaron en las rutinas MOMAP:63.241:665, en la que se decidía la estrategia de visita de pantallas, y MOPAN:57.704:859, en la que se recorría la pantalla en concreto para intentar conseguir el objetivo puntual (llegar a una salida o recoger un objeto). No se proporcionó ayuda extra al personaje automático (como darle a conocer la ubicación de los cristales), por lo que el efecto final que se obtuvo fue bastante parecido al de un jugador humano.



Fragmento de pseudocódigo para la IA

Originalmente comenzamos el ensamblado en la dirección 32.768, pero esto ya no fue una opción cuando el juego fue creciendo. Recuerdo aprovechar cada byte libre de memoria RAM (en los papeles se pueden leer anotaciones como “Aquí sobra un byte”). Dado que los primeros 16KB estaban reservados a la ROM y los siguientes 6.912 bytes a la memoria de video, disponíamos a partir de la dirección 23.296 para trabajar (42.240

bytes), cuidando de no reutilizar ciertas direcciones de las primeras posiciones de la RAM libre que usaban las rutinas de la ROM a las que llamábamos. La práctica totalidad de los gráficos y el mapa se ubicaron en estas primeras posiciones de memoria, y a partir de la posición 36.575 comenzaban las variables globales y el código del programa.

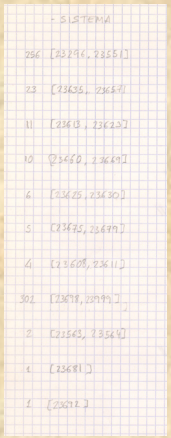
El diseño del mapa de memoria del juego incluyendo la ubicación de gráficos, variables y rutinas fue un proceso absolutamente manual. Tenía que ser consciente de lo ocupado por cada pieza del puzzle y la dirección en que lo podíamos ubicar. Elaboraba una lista de referencias internas y de llamadas a otras rutinas con sus direcciones absolutas de memoria, y procuraba tener referencias relativas para los puntos de entrada secundarios para simplificar un poco el proceso, aunque esto consumía valiosos bytes. Si una rutina crecía demasiado debía rehacer el mapa manualmente y reubicar todos los archivos afectados. Si se reducía, a veces era necesario hacer lo mismo y en otras ocasiones simplemente lo aprovechaba para alojar variables globales. Por supuesto, no era razonable gestionar esto con *Spectrum*, dado que era la máquina de desarrollo y la multitarea aún no estaba implementada.

Variables de memoria y rutinas con sus referencias internas y externas

Para reducir los requisitos de memoria en la versión final, sobrescribíamos el cargador de cinta específico y el juego de marcianitos con código de nuestro juego llamando a la rutina de carga normal al final de la específica. En esta carga final se almacenaban en particular los gráficos del panel de estado en el tercio inferior de la pantalla, aunque con atributos de color negros para mostrarlo sólo cuando se iniciase el juego. Por el mismo motivo de ahorro también recurría en ocasiones al segundo juego de registros de la CPU, pero al conmutar de uno a otro perdía el acceso al anterior y esto limitaba las aplicaciones. Al final del desarrollo sobraron unos pocos bytes que quizás se pueden contar con los dedos de las manos. Para preservar nuestra autoría dejamos nuestros nombres codificados a lo largo del código, además de en algún gráfico.

Nuestros nombres en algún gráfico

Durante el desarrollo, el ensamblador ocupaba distintas direcciones de la compactación final. Por ejemplo, el ensamblador ZEUS se ubicaba en la zona superior de memoria, a partir de la dirección 57.344, y sus fuentes a partir de la dirección 32.768. Mis variables y código ensamblado a partir de la 42.500 (un valor al que llegué después de extenderlo en más de una ocasión cuando los fuentes crecían demasiado). También tenía anotada una lista de posiciones libres en el espacio RAM usado por las rutinas de la ROM que podía aprovechar. En esta fase de desarrollo no cargaba la mayoría de los gráficos para hacer hueco al ensamblador, los fuentes y las variables de la ROM.



Algunas posiciones candidatas en RAM usadas por rutinas de la ROM

Finale

El desarrollo de *Vega Solaris* fue arduo, complejo y extenso, pero a la vez gratamente edificante. Creo que la disciplina y el método que desarrollé al llevarlo a cabo fueron piezas realmente útiles de las que me he valido a lo largo de mi carrera. Y entender el por qué de su génesis y consecución sólo se puede explicar porque existen emociones como la curiosidad, la ilusión, el asombro, el afán de superación y el orgullo. Algo que todos, en mayor o menor medida, llevamos forjado en nuestra impronta.

Finalmente me gustaría agradecer especialmente a mi familia haber puesto los medios necesarios para llevarlo a cabo, a todos los que han ayudado y colaborado en la recuperación y preservación de la versión final, así como todos los comentarios de afecto y apoyo recibidos. En la página web http://www.fdi.ucm.es/profesor/fernan/pg/html/vega_solaris.html he recopilado una buena cantidad de material sobre el juego para los *curiosos*.