NÉMESIS

“Un sistema multiplataforma para la creación de aventuras conversacionales”

VERSIÓN 1.01

Distribución de Némesis

Némesis está distribuido bajo la licencia general pública de GNU, GPL. Puedes encontrar esta licencia completa en el fichero GNULICENSE.

Básicamente, Némesis es gratuito, pero está protegido por la licencia GPL. Puedes copiar y utlizar Némesis líbremente sin cargo alguno.

Las fuentes de Némesis están incluidas. Tienes derecho a modificar estas fuentes y a distribuirlas por tu cuenta, siempre informando de la procedencia original.

Importante: como los juegos creados con Némesis usan parte de su código fuente (varios ficheros .C) protegidos por GPL, **estás obligado a distribuir estos ficheros junto a los ejecutables tu aventura**. Es decir, cuando distribuyas tus juegos, debes incluir las fuentes .C, tanto las canónicas (pronto sabrás qué son) como las generadas por Némesis. No así la fuente Némesis de tu aventura.

Qué es Némesis

Némesis es un novedoso sistema para la creación de aventuras conversacionales. Si no sabes lo que es una aventura conversacional, no necesitas Némesis, aunque te recomendaría que pasaras por http://www.geocities.com/zforzero para informarte sobre ellas.

¿Qué ofrece Némesis frente a otros sistemas de desarrollo?

Características de Némesis:

- sistema de desarrollo multiplataforma: Némesis debería estar disponible en el momento en que leas esto para las siguientes plataformas: Linux (entorno nativo de Némesis), Windows 32, MS-DOS y Solaris (para procesadores UltraSparc). Los juegos desarrollados con Némesis además deben compilar en todas estas plataformas sin modificaciones.

- generación de código C: Némesis genera fuentes en ANSI C con el juego que has creado. Esto proporciona la multiplataforma, pues es posible compilarlas en cualquier sistema. Esto además permite superar la potencia de cualquier otro parser existente pues te permite incorporar C directamente en tus programas.

Acerca del autor de Némesis

Némesis ha sido creado por *Javier Basilio Pérez Ramas* (Jaba).

No es el primer parser que he creado. Desde 1988, he venido trabajando en otra herramienta llamada DAP (Diseñador de Aventuras Profesional), que posiblemente todavía siga actualizando. Si realmente te asustan los ficheros en lenguaje C, o consideras Némesis demasiado complejo, tal vez estés interesado en DAP. Aunque DAP no tiene las potentes características de Némesis, es un sistema tremendamente fácil de utilizar que no requiere nada de programación. Visita la página del proyecto DAP en http://www.geocities.com/erjaba/dap.htm

Para contactar conmigo (realmente me gustaría saber cualquier opinión que tengas que darme sobre el programa):

email: erjaba@teleline.es
página personal: http://www.geocities.com/erjaba
irc: habitual (con el nick _Jarlaxle) en los canales #caad, #boqueron, #multiplexor, #Amstrad, #spectrum y #menzoberranzan.

Saludos

Aprovecho este documento para saludar a algunos individuos del panorama aventurero nacional...

Fran Morell, David Miraut, Javier San José, Zak, Sirrus, mk360, Sergio Llata, Daniel Cárdenas, Ramón Fusté, Jaydee, Gonzalo Flores, dhan, chir, y perdón por todos los que me dejo en el tintero...

Caracteristicas actuales

Versión 1.00 – Lanzada el 30/10/2000

- potente sistema de desarrollo de aventuras
- lenguaje de construcción de rutinas de alto nivel y de alta potencia.
- generación de código C.
- genera programas con un rendimiento tremendamente superior al de otros sistemas.
- no hay límite de mensajes, localidades, etc...
- soporte de guardar y recuperar el estado de juego

Historia

1.0 Alpha pública 7/2000 – primera versión
1.0 (30/10/2000)
- añadido save y load
- añadido flag de nocogible
- añadidos los flags de objetos
- añadidas nuevas funciones para acceder a los flags nuevos, así como para ver si una cadena está contenida en otra.

1.01 (18/12/2000)
- documentación corregida

Lo que viene

Algún día que quizás no llegue, Némesis contará con:

- una documentación más presentable que ésta
- gráficos usando GLK
- entorno de desarrollo de aventuras (IDE) escrito en TCL/TK, de modo que sea migrable a distintas plataformas.
- versión Windows con compilador C integrado

Instalando Némesis

VERSIONES LINUX Y SOLARIS

Una vez descomprimido el gzip y "des-tar-eado" puedes ejecutar el script instalar ("source instalar"). Este script instala Némesis en tu directorio personal, en el directorio nemesis.

VERSIÓN WINDOWS

Descomprime el fichero ZIP en el directorio que desees.

Comenzando

Un juego creado en Némesis se divide en tres partes: la definición de localidades, la definición de objetos y la definición de verbos. Cada una debe estar en un fichero de texto distinto.

Nota: supongo que tienes cierto conocimiento del uso de tu sistema LINUX, Windows, etc…

Para crear una aventura debes ir al directorio de Némesis y ejecutar el script (o fichero BAT) nueva seguido el nombre de la misma. Por ejemplo:

C:\NEMESIS\>nueva prueba

Esto creará un subdirectorio llamado "prueba" con todo lo necesario. Entra en el directorio. Allí debes crear los tres ficheros de la aventura. Cada fichero debe tener el nombre de la aventura y, como extensión .LOC, .OBJ o .PRG según se trate de la definición de localidades, objetos o verbos. Estos ficheros deberás crearlos con tu editor de texto favorito.

En todos los ficheros puedes poner comentarios encerrados entre /* y */

<u>El fichero de localidades</u>

Este fichero contiene las localidades del juego y sus conexiones.

El formato del fichero es el siguiente:

max=NUMERO;

Debe ser siempre la primera línea, e indica el número máximo de localidades de la aventura. No es necesario que luego crees todas las localidades especificadas en NÚMERO; sólo establece un máximo para el compilador. He llegado a compilar aventuras con más de 10000 localidades sin ningún problema bajo Linux. Por favor, observa que aunque no emplees todas las localidades que has definido SÍ ocupan memoria éstas.

Después, una a una, debe ir la definición de cada localidad, por ejemplo:

```
loc 1
‘ Éste es el jardín de la casa. La vieja mansión domina el misterioso ambiente.\nUn halo de misterio envuelve el lugar… La puerta está cerrada.’;
aka=ENTRADA; /* esto permite referirse a la localidad por ese nombre en lugar de por su nómero */
finloc;

loc 2
‘Estás dentro de la casa. Las escaleras permiten subir a un piso superior.\nEl mobiliario es antiguo y lujoso.\nAl este hay una puerta cerrada.\n’;
sur=1;
aka=RECIBIDOR;
finloc;

loc 3
‘Estás en el piso de arriba.’;
aka=PISOARRIBA;
finloc;

loc 4
‘Estás en la cocina.’;
aka=COCINA;
oeste=2;
finloc;
```

Observa el formato en que se ha definido cada localidad:

Primero se pone loc y el número de localidad.

Después siempre viene la descripción de la misma, encerrada entre comillas simples. Puedes incluir el símbolo \n para saltar de línea.

Después vienen los atributos de la localidad. Son lineas de la forma atributo=valor. Los atributos son:

- las salidas: norte, sur, este y oeste. El valor debe ser la localidad por la que se conecta con esa salida. Si la salida es 0, la localidad no tiene conexión por esa dirección. Si omites alguna salida, se supone 0. Las localidades pueden contar con otras salidas del estilo entrar, salir, subir … pero no se soportan directamente sino que hay que implementarlo en Nembasic.
- los AKAS: cada localidad puede tener AKAs. Cada AKA da un nombre simbólico a la localidad (p.ej, RECIBIDOR a la localidad 2). Esto permitirá referirse a la localidad por el nombre AKA durante la programación en lugar de por su número, ahorrándote tener que recordar el número de cada localidad. LOS NOMBRES DE AKA DEBEN ESTAR EN MAYÚSCULAS.

Finalmente, cada localidad acaba por la palabra **finloc.**

No es necesario definir las localidades en orden, aunque por supuesto es recomendable.

El fichero de objetos

Primero una somera descripción de cómo Némesis trata a los objetos …

En Némesis, cada objeto tiene un estado que puede valer:

jugador ⇨ el jugador está en posesión del objeto.
Limbo ⇨ el objeto está en un "limbo", es decir, en ninguna localidad y tampoco lo tiene el jugador.
Localidad ⇨ el objeto está en una localidad.

En el tercer caso, además, un objeto contiene un número de localidad.

Además, cada objeto tiene los siguientes atributos:

- descripción: describe el objeto al jugador. Se mostrará cuando el parser diga los objetos que hay en cada localidad, y también será el nombre que se de al objeto en el inventario del jugador. Cuando el jugador coja o deje el objeto deberá especificar su nombre. La descripción puede cambiarse durante la ejecución de la aventura.
- nocogible: es un flag que indica si el objeto puede cogerse o no. Si este flag existe, el objeto no puede cogerse. Su principal uso es la creación de PSIs, que simulan personajes no jugadores; Némesis los trata como un tipo especial de objeto que no puede cogerse.
- flags: cada objeto contiene 15 flags de valor entero de uso no especificado. Como programador puedes usarlos de la forma que más te convenga; por ejemplo, para especificar lo que pesa el objeto (lo cual te obligaría a definir una rutina de coger a medida, más sobre esto adelante) o la carga que le queda a una pistola…

El fichero de definición de objetos es similar al de localidades.

Primero se pone una línea:

maxjug=n;

Donde n es el máximo número de objetos que podrá llevar el jugador.

Después viene la línea:

max=n;

Que es el número máximo de objetos que admitirá la aventura; como en las localidades, no es necesario definir todos sin embargo.

Con un ejemplo vemos la definición de los objetos:

```
/* OBJETOS PARA LA AVENTURA DE EJEMPLO */

maxjug=2; /* Máximo que puede llevar el jugador */
max=3; /* Máximo de objetos admitidos */


obj 1
‘una llave’;
aka=LLAVE; /* esto permite referirse a la localidad por ese nombre en lugar de por su nómero */
estado=limbo;
finobj;

obj 2
‘tu DNI’;
aka=DNI;
estado=jugador;
finobj;

obj 3
‘una flor’;
aka=FLOR;
nocogible;
estado=limbo;
finobj;
```

Para cada objeto ponemos, en primer lugar, la descripción, y después los atributos. Los atributos de un objeto son:

aka=NOMBRE;

Que tienen el mismo propósito que en las localides, es decir, por ejemplo, a lo largo del programa podremos referirnos a la flor como FLOR en lugar de como “objeto número 2”. Los AKAS deben ESTAR EN MAYÚSCULAS.

Estado=e;

el estado: limbo, jugador o localidad.

Localidad=n;
Loc=n;

indican la localidad inicial del objeto (son equivalentes). Sólo tienen sentido para el estado localidad, aunque no es un error incluirlas en otro caso.

nocogible indica que no se puede coger el objeto. Es útil para definir PSIs.

No es posible dar valores iniciales a los flags de un objeto aquí, sino que habrá que hacerlos sobreponiendo la función inicio de Nembasic (ver esto más adelante). Además, al empezar la aventura los flags tienen valores no determinados, no puede suponerse que estén a cero.

Creando la aventura

Bien. Ya tienes creada la estructura de tu aventura, y estás listo para compilarla y ejecutarla. Crea un fichero PRUEBA.PRG vacio, de momento no vas a definir verbos.

De esto se encarga el script (o fichero .BAT) “cm”

Ejecuta cm, dentro del directorio de la aventura, con el nombre de la aventura como parámetro:

yo/nemesis/prueba% nueva prueba

Verás los mensajes de Némesis. Si hay errores, se informará de ellos y en qué línea están.

NOTA: en algunos casos, puede ocurrir que Némesis no sea capaz de capturar el error y lo haga el compilador de C, siendo esto más difícil de identificar. Estoy trabajando para que Némesis pueda detectar todos los errores.

Si todo el proceso ha transcurrido sin errores, la aventura ya está lista en el fichero ejecutable PRUEBA. Ejecútalo y prueba tu aventura. Tendrás disponibles las órdenes básicas: norte, sur, este, oeste, coger … que son incorporadas automáticamente por Némesis.

PROGRAMANDO LA AVENTURA

Es necesario, al crear una aventura, especificar a Némesis qué quieres que suceda ante cada acción del jugador. Esto se hace asociando una rutina a cada acción posible. Por ejemplo, cuando el jugador “abra” algo, se ejecutará una rutina asociada al verbo “abrir”.

Las rutinas se definen en el fichero con extensión .PRG. Para ello, se utiliza un lenguaje sencillo de programación de alto nivel, NemBasic. Con este lenguaje tienes la potencia para hacer todo lo que se te pueda ocurrir de forma rápida y cómoda.

La forma general del fichero PRG es la siguiente:

verbo v;
  cuerpo

etc…

Así para definir cada verbo que entenderá el parser.

V es el verbo que dispara la rutina. “Cuerpo” es la rutina propiamente dicha, escrita en NemBasic.

Se admite la palabra ‘sub’ en lugar de ‘verbo’.

El lenguaje NemBasic

Este manual no pretende ser una guía de programación, así que describiré rápidamente NemBasic y dejo para una revisión posterior un manual mas detallado.

Observa que detrás de cada orden de NemBasic debe ir un punto y coma.

Tipos de datos:

NemBasic soporta dos tipos de datos: entero y cadena de caracteres. En las operaciones lógicas se usan enteros, de forma parecida a C. Un valor entero distinto que cero es cierto, y uno igual a cero es falso. El tamaño del entero depende del sistema operativo, normalmente será de 32 bits.

Variables:

Deben declararse justo después de definir el verbo, con el tipo por delante. Por ejemplo:

```
verbo abrir;
  entero v1;
  cadena v2;
  entero v3;
```

no es posible dar valores durante la declaración, ni dar una lista de variables tras un tipo.

Imprime

Imprime el valor especificado. Éste puede ser un entero o una cadena

Por ejemplo:

```
imprime ‘Hola’;
imprime 2+3;              (imprime 5)
imprime ‘mocos’+’elefantes’; (imprime “mocoselefantes”)
imprime 5+’monos’; (imprime “5monos”)
imprime boqueron; (imprime el valor de la variable boqueron)
```

Observa que el operador + se puede usar para concatenar cadenas o cadenas con números.

Némesis, para el trabajo con números, admite los operadores +-,*,/,(y). Además se respeta la prioridad algebraica.

Estructuras de control

si (condición) instrucciones fsi;

Ejecuta las instrucciones si se cumple la condición.

Para comparaciones lógicas, se admiten los operadores =,<>,< y >. Se pueden aplicar tanto a enteros como a cadenas. Némesis realiza las conversiones de tipos adecuadas.

Se admiten los operadores “y” y “oo” para la AND y la OR lógicas.

Aprovecho este punto para señalar que desde un verbo siempre se puede acceder a una variable llamada **nombre**, que contiene el nombre que ha obtenido el parser de la entrada del jugador.

```
verbo abrir;

  si nombre=’puerta’
        imprime ‘Abres la puerta’
  fsi;
```

si (condición) instrucciones sino instrucciones fsi;

Como la anterior, pero sino se cumple la condición ejecuta el segundo bloque de instrucciones.

```
verbo abrir;

si nombre=’puerta’
  imprime ‘Abres la puerta’;
```

```
sino
  imprime 'No sé que pretendes abrir'.
fsi;
```

Asignación

Modifica el valor de una variable:

```
suma=suma+3;
suma=suma*(5+10)-suma;
frase='Tengo '+cuantos+' boquerones'.
```

Se admiten, en general para cualquier expresión, los operadores +,-,* y /. Sólo el primero puede aplicarse a cadenas, actuando en ese caso como concatenador. Además, sumar una cadena con un número siempre produce una cadena.

Además se admite el operador **no** lógico, el - unario (negación), y los paréntesis:

```
var=-var;
var=-5*(2+4*5);
```

Por supuesto, la evaluación se realiza en el orden aritmético esperado: los operadores de multiplicación tienen preferencia sobre los aditivos, etc…

Las constantes del sistema "cierto" y "falso" equivalen a 1 y 0, respectivamente.

Bucle mientras

mientras (condicion) instrucciones fmientras;

Repite las instrucciones mientras se cumpla la condición.

```
verbo cosa;

entero a;

a=1;
mientras a<11
  imprime a+' elefantes se balanceaban…';
  a=a+1;
fmientras;
```

Llamada a subrutina - Creación de sinónimos

Se puede llamar a un verbo desde otro. Esto permite crear fácilmente subrutinas y sinónimos.

Se hace con “usa verbo”:

```
verbo coge;
  usa coger; /* crea un sinónimo de coger */
```

El verbo llamado comparte el mismo nombre que el llamador.

Como no se admite el paso de parámetros, si deseas hacerlo en subrutinas tendrás que usar variables globales para comunicarlas. Lo malo de esto es que no permite la recursión, pero no veo en que puede ayudar ésta en una aventura conversacional.

Ret

Finaliza la ejecución y vuelve al bucle principal de la aventura.

Quitar

Finaliza la ejecución de la aventura.

Dos funciones primitivas adicionales permiten leer datos del teclado:

lee_entero() ⇨ es una función que lee un entero y devuelve su valor
lee_cadena() ⇨ análogo con una cadena de caracteres

P.ej:

```
  sub prueba;
        cadena nombre;

        imprime ‘Dime tu nombre’;
        nombre=lee_cadena();
        imprime ‘Hola ‘+nombre;
```

A continuación se describen instrucciones que permiten acceder y modificar datos de la aventura:

Modificaciones de localidades

pon_salida(salida,nuevasalida,loc)

Modifica la salida especificada de la localidad indicada, haciendo que vaya a la localidad nuevasalida.

Ej: pon_salida(sur,3,HABITACION);

nota: HABITACION es un AKA definido en el fichero de localidades.

pon_desc(descrip,loc)

Modifica la descripción de una localidad.

Ej: pon_desc('La habitación está indundada',HABITACION);

Modificaciones de objetos

obj_a_jug(num);

Hace que el objeto número num pase a estar en posesión del jugador.
Si el jugador ya no puede llevar más objetos no tiene ningún efecto.

obj_a_loc(objeto,loc)

Hace que el objeto con el número especificado pase a la localidad indicada. Si el jugador tenía el objeto, lo pierde.

obj_a_limbo(objeto)

Hace que el objeto con el número indicado pase al "limbo" de objetos. Si el jugador lo tenía, lo pierde.

pon_obj_flag(objeto,numeroflag,valor)

Da un nuevo valor al flag especificado de un cierto objeto. Los flags son 16, de 0 a 15. Acceder a uno más alto no dará un error de compilación pero tendrá consecuencias indeseadas para la aventura.

FUNCIONES

Estas son funciones que proporcionan un valor que puede usarse dentro de una expresión. Las funciones reciben argumentos y devuelven o bien un entero o una cadena.

Funciones referentes a localidades

lee_salida(salida,loc)

Devuelve la localidad a la que apunta la salida de LOC**.** P.ej, lee_salida(este,CASA). Si no hay salida, devuelve 0.

lee_desc(loc)

Devuelve el valor de la descripción de esa localidad.

Por ejemplo: imprime(lee_desc(CASA));
Imprimiría la descripción de esa localidad.

Funciones referentes a objetos

obj_num(descripción)

Esta función es muy útil. Recibe una cadena y devuelve el número del objeto con la descripción más similar. Esto evita tener que programar usando los números de objetos e incluso los AKAs de objetos. Si no existe ningún objeto que ajuste, vale 0.

obj_tiene(num)

Devuelve un valor cierto o falso que indica si el objeto con el número especificado está en posesión del jugador.

obj_esta(obj,loc)

Dice si el objeto con el número obj está en la localidad número loc.

obj_limbo(obj)

Dice si el objeto con ese número está en el limbo de objetos.

Lleno()

Dice si el jugador tiene el máximo número de objetos que se le permiten.

obj_cogible(obj)

Dice si ese objeto se puede coger.

lee_obj_flag(obj,numflag)

Devuelve el valor del flag deseado de un objeto.

Otras funciones

cadenacontenida(cad1,cad2)

Devuelve un valor lógico que indica si la cadena cad2 forma parte de la cadena cad1. Es útil para definir un verbo “decir” y hablar con psis …

VARIABLES GLOBALES

Se pueden definir variables globales, que son variables que pueden accederse desde todos los verbos. Esto es útil para crear banderas o indicadores del estado de juego: si cierta puerta se abrió, etc…

Estas variables se deben declarar al principio del fichero:

```
global entero ventanarota;
global entero pasoporalli;
```

Además, cuando se use una variable global dentro de un verbo, debe declararse al principio de la misma forma.

NÉMESIS además ofrece algunas variables globales predefinidas que pueden accederse si se declaran dentro de cada verbo:

gloc ⇨ localidad actual del jugador. Observa que modificando su valor puedes mover al jugador de localidad.
MAXJUG ⇨ máximo número de objetos que puede llevar el jugador.
MAXOBJ ⇨ máximo de objetos existentes.

Observa que Némesis distingue entre mayúsculas y minúsculas.

VERBOS ESTÁNDAR

NÉMESIS predefine algunos verbos estándar que suelen existir en todas las aventuras y, por tanto, no es necesario que definas:

coger (objeto)
dejar (objeto)
inventario
norte
sur
este

oeste
save
load

Sin embargo, puedes redefinir estos verbos, con lo que la rutina estándar de Némesis quedará "sobrepuesta". Éste es un proceso llamado "sobreposición" de funciones. Una excepción son save y load, que no es posible redefinir.

Si defines por ejemplo, el verbo coger, la rutina coger estándar de Némesis queda desactivada y la tuya ocupará su lugar. Esto permite personalizar al máximo tus aventuras. Pero si quieres crear un juego rápido o no quieres perder el tiempo creando estas rutinas (algo complejas) Némesis te ofrece el juego estándar.

Ejemplo: sobreposición del verbo inventario:

```
verbo coger;

  global entero gloc;
  entero num;

  num=obj_num(nombre); /* obten el número de objeto */


  si num=0
        imprime 'No sé que es eso'; /* objeto que no ajusta a ninguno */
  fsi;

  si obj_esta(num,gloc) /* mira que esté ahí */
        obj_a_jug(num); /* recógerlo */
        imprime 'Lo coges';
  sino
        imprime 'Por aquí no está';
  fsi;
```

Además existen algunas funciones de Némesis que no son verbos pero que son fundamentales para la aventura. También puedes sobreponer estas funciones, de lo contrario, Némesis añade unas por defecto:

describir ⇨ se llama cada vez que el jugador entra a una localidad. La función por defecto muestra la descripción de la localidad y los objetos que hay en ella.
inicio ⇨ se llama al empezar la aventura. La función por defecto no hace nada.
central ⇨ se llama después de cada turno. La función por defecto no hace nada.

Observa que si sobrepones coger deberás comprobar que el objeto es cogible y que el inventario del jugador no está lleno. No des nada por sentado al sobreponer funciones.

Algunos aspectos de más bajo nivel

Némesis, para toda la entrada y salida, usa unas funciones llamadas nemputchar(), nemputs() y nemgets() contenidas en “nemio.c”. Si quieres modificar la forma en que se realiza la salida, puedes modificar estas funciones.