

Si ya está familiarizado con lenguajes de programación y scripting como JavaScript o C++, te puede saltar este capítulo y empieza por la lección del lunes sobre las puertas y llaves. Si no es así: inicia el Wed, y abre y ejecuta el nivel "office", (pulsa [run], y no te olvides antes de [build]). mientras mirando el cielo, pulsa la tecla [Tab]. El juego se congela, y un cursor parpadeante aparece en la pantalla.

Ahora escribe.

```
sky_scale = 0,3;
```

y pulsa [enter]. Si ahora observas un cambio en la textura del cielo, sólo fuiste testigo de los efectos de tu primera instrucción de script. Modifica el 0,3 a otro valor utilizando las teclas del cursor y [Del], a continuación, presiona [enter] de nuevo. Jugar con los valores y observar el efecto en el juego. Puede salir de este modo de entrada directa a través de [Esc].

Lo que has hecho es cambiar el valor de una variable llamada sky_scale. No te sorprendas, esta variable es la responsable de la ampliación de la textura del cielo en nuestro nivel "office". Las intrucciones Script son sólo texto que puede ser mecanografiado en un editor de texto, o - utilizando la tecla [Tab] - en el mismo juego. El primero es el método normal, el segundo es para las pruebas y jugar. Por cierto, la tecla [Tab] es la que permite escribir datos en una línea del WDL.

Salir del juego, y ahora abrir el archivo office.wdl en el directorio de trabajo utilizando un simple editor de texto como -Windows™ Bloc de notas- por ejemplo. Como puedes ver, los archivos de script deben terminar con la extensión WDL. La mejor solución de escribir tus scripts es el editor WDL, que lo puedes descargar en la pagina de enlace de Gamestudio. Nunca utilices un procesador de textos como Word o WordPerfect™ para los scripts. los archivos con sus códigos de formato no son validos para el compilador de c script. puedes ver ahora el comienzo del archivo office.wdl, que se parece a esto (o similar):

////////////////////////////////////
// Office test level
////////////////////////////////////
path "..\\template"; // ruta de la carpeta de trabajo a la carpeta de plantillas
include <movement.wdl>; // libraries of WDL functions, located in TEMPLATE
include <messages.wdl>;
include <doors.wdl>;
...
////////////////////////////////////
// After engine start, the MAIN function is executed
...
function main()
{
...

Esta es una secuencia de comandos básicos, que son creados automáticamente tras hacer clic en [new] en el Botón Map Properties.

Todo entre // y el final de una línea es un comentario y será ignorado por el motor.

Una regla básica de buena secuencias de comandos es que siempre debes pensar en la próxima persona que tiene que mirar a tu script. Podría ser un amigo, un compañero de trabajo, un empresario, o podría ser tu dentro de tres meses. La forma más fácil de asegurarse de que tu comprendas tu propio código dentro de tres meses es comentarlo con frecuencia. Si desea comentar un enorme bloque de texto, puede ponerlo entre /* y */ como esto:

```
/* this is a block  
of text that I've  
commented out */
```

Al comienzo de una secuencia de comandos de un archivo se da una ruta de directorios, a muchos más archivos de script que se incluirán en el script principal. "Include" significa que se manejan como si su contenido estuviese directamente escritos en este script principal. Estos archivos "plantilla" contienen una «biblioteca» de funciones predefinidas y deben estar disponibles en el juego (en la misma carpeta). Estos se describen en otros capítulos del manual, y puede ser utilizados para hacer juegos simples con muy pocas líneas de programación.

La función principal que figura a continuación es el programa real. Es ejecutado directamente en el arranque del motor, y contiene todas las cosas que se haran en el comienzo del juego que es, en el ejemplo, la carga del nivel office.wmb.

Sin embargo, ahora vamos a comenzar con el aprendizaje del lenguaje - la forma en C-Script almacena la información, cómo se toman decisiones basadas en esa información, y cómo cambiar la información basada en la interacción del usuario. ¿Listo? Es hora de aprender los fundamentos de la programación de computadoras. Primera parada: variables.

Variables y Strings

Si ha aprendido álgebra, sabes que son variables. Si no, no te preocupes. Las variables son simplemente la manera en que cualquier lenguaje de programación almacena la información numérica. Por ejemplo, si escribe "x = 2", "x" es una variable que tiene el valor "2". Si luego decimos "y = x+3", "Y" tomará el valor "5". He aquí un ejemplo de secuencia de comandos que crea las variables:

// define some vars
var secs_per_min = 60;
var mins_per_hour = 60;
var hours_per_day = 24;
var days_per_year = 365.25;
var secs_per_day;
var secs_per_year;

Como puede ver, cada secuencia de comandos de declaración o instrucción debe terminar con un punto y coma. En caso de que la olvide, puedes estar seguro que recibirás un mensaje de error del motor.

La primera línea que comienza con '// ' es un comentario de que definimos variables. El siguiente grupo de líneas son definiciones de variables. Hay unas cosas que comentar acerca de estas líneas: Antes de poder utilizar una variable, se deben definir con la palabra "var", seguida por su nombre y, a continuación, por un facultativo valor inicial después de un signo "=". Algunas variables (como nuestro sky_scale) no tienen por qué ser definidas ya están predefinidos internamente por el motor. A diferencia de su 'hermano mayor' C++, C Script conoce sólo un tipo de variables.

C++ tiene docenas de tipos de variables para diferentes fines, como int, float, doble, corto, largo, bool ... en C-Script sólo se puede utilizar "var".

Los nombres de variables deben empezar por una letra o el carácter de subrayado. Después del primer carácter, las variables pueden tener números. Harry_23 Por lo tanto, es un nombre aceptado para una variable. En los nombres de variables c script no distingue entre mayúsculas y minúsculas. Esto significa que las variables loop, y LOOP se considerará la misma. En general, es una buena idea para no confundir variables y vamos a atenernos a ella. A mi me gusta poner todas mis variables minúsculas, separando las palabras con un guion bajo. Otra gente prefiere usar la nominación interna, como SecsPerMin. Las variables deben describir lo que son. Nombres de variables tales como A, B, o un número, no son fáciles de identificar a una persona que está tratando de estudiar tu script. No hagas que tus variables, tengan siempre un nombre largo, pero si que sea lo suficientemente largo como para ser descriptivo.

Tu puede dar a una variable un valor inicial cuando se definen. En el ejemplo, a algunas de las variables se les dio un valor la primera vez que se definieron. no siempre tienes que hacer esto, y veremos ejemplos en los que es bueno para definir una variable, aunque no sepamosmos su valor de inmediato.

Una variable puede representar más de un número (más tarde aprenderás

acerca de los vectores que contienen tres números, y arrays que contienen la cantidad de números que deseas). Las instrucciones finalizan con un punto y coma. Las instrucciones son las frases en un lenguaje script y punto y coma al final son marcas de puntuación. Los espacios y saltos de línea normalmente se les hace caso omiso, por lo que en el diseño de la secuencia de comandos sólo sirve para hacerlo más legible para otras personas. Este ejemplo podría haber sido escrito en una línea muy larga si se toman los comentarios. Sin embargo, sería difícil de leer.

Funciones

¿Listo para hacer algo con nuestras variables? Un objeto que hace algo con algo es una función, que consta de instrucciones:

```
// do some calculations
function some_calculations()
{
secs_per_day = secs_per_min * mins_per_hour * hours_per_day;
secs_per_year = secs_per_day * days_per_year;
}
```

Las reglas de nomenclatura de funciones son las mismas que para las variables. El primer carácter tiene que ser una letra o un guión bajo. El resto pueden ser números. También, debes asegurarte de que no nombrar una función con el mismo nombre de una variable. El nombre es seguido por dos paréntesis, entre los que van los parámetros "los números, por ejemplo" que podrían ser transferidos a la función. Pero no estamos utilizando parámetros de aquí, por lo que el paréntesis está vacío. Luego viene el cuerpo de la función dentro de las llaves. Este es el conjunto de instrucciones que deseas ejecutar cuando se ejecuta la función. En nuestro ejemplo, vamos a ver dos con algunas instrucciones básicas de matemáticas.

La expresión de la derecha de '=' se calcula, y el resultado es colocado en la variable de la izquierda de '='. Por lo tanto, después de que el programa ha ejecutado la función, la variable `secs_per_year` contendrá el resultado de $60 \times 60 \times 24 \times 365,25$. A partir del momento en que la función es ejecutada, siempre que el programa considera la variable `secs_per_year`, se sustituirá por este resultado.

Puedes aprender otra cosa de esta función: En la mayoría de los casos, el orden de las instrucciones tiene que seguir una pauta. Las instrucciones se ejecutan desde el comienzo de la función hasta el final. Si tuviéramos intercambiadas las dos instrucciones, el resultado sería absurdo, porque la segunda instrucción necesita `secs_per_day` ya calculado. Después de lo que hemos hecho con las variables, ahora vamos a introducir un segundo tipo de objeto para

almacenar información, las cadenas o string. Las cadenas son algo similar a las variables, pero que contienen caracteres en lugar de números. Cualquier grupo de caracteres, entre comillas, es válido para el contenido de una cadena. Por lo tanto, es legal definir:

```
// Definición de una cadena string hello = "Hello World!";
```

Escribir esta línea en un archivo de comandos define la cadena hola y la llena con los caracteres entre comillas. Cadenas y variables son similares, pero no se pueden hacer matemáticas con una cadena, pero puedes cambiar el contenido de la cadena como puedes con una variable.

Basta ya de la teoría. ¡Prepárate para escribir tu primer programa.

Mi Primer Programa en c script

Inicia el WED. Crea un nuevo proyecto vacío (File-> New), y guardalo (File-> Save as) nombrándolo "tutorial", a continuación, crea un nuevo archivo de script mediante la apertura de (File-> Map Properties) y haz clic en el boton[New].

El recién creado script tutorial.wdl nos indica ahora la secuencia de comandos sobre el terreno. Ahora abre tutorial.wdl en tu carpeta de trabajo con el Bloc de notas u otro editor de texto. Esta secuencia de comandos ya contiene un montón de cosas, similar al fichero office.wdl que hemos examinado antes. Borra todo -no vamos a hacer todo un juego, sólo tendremos que hacer algunos ejercicios. Introduce lo siguiente en el ahora vacío archivo de script:

////////////////////////////////////		
// some exercises		
string hello = "hello world!";		
text screen_txt // text object for displaying our strings		
{		
	font = _a4font; // standard font	
	pos_x = 5;	// text begins at (5,40) screen position (in pixels)
	pos_y = 40;	
}		
function print(str) // display a string parameter on the screen		
{		
screen_txt.string = str;		
screen_txt.visible = on;		
}		
function main()		
{		

<code>screen_color.blue = 128;</code>	<code>// set a dark blue background</code>	
<code>print(hello);</code>	<code>// display the string</code>	

Ahora guardalo, haz clic en boton [Run] del compilador del WED, [Go!]. Si no da comienzo, pero da un mensaje de error en lugar iniciarse, probablemente has olvidado una coma o algo similar falta. La línea en cuestión se muestra en el mensaje de error, para que puedas corregir tu error. Comprueba cuidadosamente, hasta que el motor arranque el script. Nuestro primer programa en C-Script se está ejecutando. Hemos programado una aplicación que simplemente muestra "Hello World!" en una ventana con un fondo azul. Bueno, no podemos hacer mucho con esto, excepto pulsar [Esc] o [F10] para salir. Sin embargo, podemos aprender mucho de este programa. Hemos utilizado un nuevo objeto, un text para mostrar cadenas de caracteres en la pantalla. Son más complicados que las variables Tienen un cuerpo encerrado entre corchetes, al igual que funciones. En el cuerpo algún defecto valores de las propiedades de texto se pueden escribir. El texto propiedades que estamos dando aquí para ver la screen_txt son la posición xy en la pantalla, y el carácter de fuente (_a4font es una fuente por defecto utilizado internamente por el motor). Siguiendo hemos definido la función de impresión. A diferencia de la función some_calculations hemos discutido antes, este toma un parámetro. Parámetros de la función son números, cadenas u otros artículos entregados a la función cuando la ejecución de la misma. El parámetro es conocido en la función con el nombre que hemos dado en el parantheses en la definición de función. En nuestro caso, usamos el como parámetro una cadena. Nuestra función de impresión muestra la cadena en la pantalla mediante el establecimiento del texto objeto de la cadena a la propiedad y, a continuación, establecer el texto del objeto visible a la propiedad sobre. Tenga en cuenta la punto después de my_text. Indica que el siguiente tema, cadena o visible, pertenece al objeto my_text. Puede leer en la referencia parte de este manual acerca de las propiedades que un texto puede tener (ir al capítulo "User Interface"). La cadena propiedad de un texto sólo da a los personajes exponer. Es visible una bandera - similar a una variable, pero sólo puede tener dos valores, encendido o apagado. Por lo tanto, es utilizarse como un interruptor de encendido / apagado, en nuestro caso para la visibilidad del objeto de texto. Cuerdas y banderas se establecen con el "=" cesión al igual que las variables. Como se mencionó antes, la función principal se ejecuta en el arranque del motor. En primer lugar, establece un fondo azul por usar la variable screen_color. Recuerde nuestra primera declaración con sky_scale? Ambos screen_color y sky_scale son variables predefinidas que un cierto cambio de propiedad ouf nuestro juego. C-Script contiene una gran cantidad de esas variables predefinidas para el control de las características de la motor. Screen_color color es un vector con los valores rojo, verde y azul, que areat0by por defecto Cada uno de los tres valores de color se puede

ajustar en el rango de 0 .. 255, y los tres juntos definir un color de la pantalla de fondo. Al fijar su valor de azul a 128, y el rojo y el verde aún en 0, estamos obteniendo un color azul oscuro. En función principal thenextlinethe ejecuta la función de impresión y las manos de la cadena hola como más parámetro. Ejecución (o llamadas) a partir de una función en otra función se realiza cuando el tipo el nombre de la función seguido por parantheses, y dentro de los parámetros (si procede). Aquí puede ver la ventaja de los parámetros de la función. Sea cual sea la cadena se pasó a la función de impresión, itwill se muestra - de manera que podamos utilizar la impresión a partir de ahora como un objetivo general la función de visualización. Tenga en cuenta que hola la cadena es conocida dentro de la función de impresión en el marco del local de nombre "str-tanto hola y str aquí son la misma cadena. Por cierto, no es necesario definir por separado las cadenas. Similar a variables, también pueden darse directamente en la instrucción, como esta:

function main()	
{	
screen_color.blue = 128;	// set a dark blue background
print("Hello world!");	// display string directly
}	

Una vez que hayas llegado a este trabajo, es el momento de aprender acerca de si las cláusulas de los demás-.

Intruccion "if"

la instrucción "if" le permite a su programa comportarse de forma muy diferente dependiendo de unas condiciones, al igual que lo que un usuario de los insumos. Esta es la forma básica de un caso de la instrucción:

if (some condition is true)
{
do something;
do something;
do something;
}

Las partes importantes de esta estructura son los siguientes:
 -Comienza con la palabra "if" (o mayúsculas "IF" si usted prefiere).
 -Hay una condición numérica en el paréntesis que sea cierto o falso.
 -Hay un conjunto de instrucciones que debe ejecutarse si la condición es verdadera. Estas instrucciones

deben ir entre llaves. Recuerde, el espaciado es sólo para que el guión sea más legible. Yo prefiero poner las instrucciones en varias líneas entre las llaves. Pero usted puede poner toda la instrucción, en una sola línea si así lo desea. Hay otra instrucción, "else", que es todo lo contrario de "if". las instrucciones entre las demás llaves se ejecutan sólo si los de la instrucción "if" no se ejecutan. He aquí un ejemplo de un caso y otra instrucción en acción:

function which_key()
{
if (key_space == 1) { // [space] key pressed?
print("Hit the space key!");
} else {
print("Hit another key!");
}
}

Es un mensaje diferente que se muestra dependiendo de si key_space es de 1 o no. Key_space predefinido es una variable que es 1 mientras que el [Espacio] clave es presionado, y de otro tipo 0.

El "==" en condición de los paréntesis es cierto, si el valor de key_space es igual a 1. Introduzca la función por encima de la función principal de nuestra tutorial.wdl, y entrar en un adicional declaración en la función principal:

function main()	
{	
screen_color.blue = 128;	// set a dark blue background
print("Hello world!");	// display string directly
on_anykey = which_key;	// assign the which_key function to the "any key" event
}	

On_anykey llama a la función asignada cada vez que una clave es golpeada. Como puede ver, hay dos maneras de ejecutar una función - o bien llamando desde dentro de otra función, o asignar a un acontecimiento que se inicia. Sólo haber escrito en un archivo de comandos no es suficiente para una función a ser ejecutado! Sólo la función principal se le asigna automáticamente al evento "Inicio del juego". Tenga en cuenta que cuando asignar una función a un evento, en lugar de llamar directamente, no se pasan parámetros, y no () paratheses se debe dar. Ahora inicie el programa, a continuación, pulse algunas teclas. Es un mensaje diferente que se muestra en función

sobre si la clave que has golpeado fue el [Espacio] clave o no. Tenga en cuenta que el "==" condición es la igualdad de dos signos. Esta es una de esas cosas que todo el mundo se mete inicialmente. Si coloca un signo igual en lugar de dos, no el trabajo - un signo igual significa un atribuidas, y no una condición. Otras condiciones típicas son: (var_1 > var_2) si es cierto var_1 es mayor que var_2 (var_1 < var_2) si es cierto var_1 es inferior a var_2 (var_2 <= var_2) si es cierto var_1 es inferior o igual a var_2 (var_2 > = var_2) es cierto var_1 si es mayor o igual a var_2 (var_1 ! var_2 =) es cierto si no var_1 igual var_2 Dos formas de hacer que las condiciones de su elegante. Si desea tanto de dos cosas para ser verdad antes de ejecuta las instrucciones de las llaves, usted puede hacer esto:

```
if ((var_1 > 18) && (var_1 < 21))
{
do something;
}
```

Notificación de los dos ampersans. Así es como usted dice "y" en un lenguaje de programación. Aviso también que la cláusula en su conjunto, incluidos los dos sub-partes y la ampersans debe ser incluida en paréntesis. Si quiere al menos una de estas dos cosas para ser verdad para ejecutar las instrucciones en el rizado entre paréntesis, siga estos pasos:

```
if ((var_1 == 7) || (var_2 == 13))
{
do something;
}
```

»¿Ok, ahora es el momento de hacer un poco de examen de las cosas que hemos leído hasta ahora. Si usted siente que no aprender una de estas cosas, volver atrás y cazar para ella: - // y /* */ se utilizan para comentarios. Comenta a menudo tus escrituras. - las Variables puede contener números. Existen algunas reglas de oro a tener en cuenta a la hora de nombrar variables. - las cadenas string puede contener secuencias de caracteres. - las funciones son para hacer o cambiar algo, y consisten en instrucciones. - las instrucciones y declaraciones deben finalizar en un punto y coma. - las funciones pueden ser llamados directamente, o asignadas a los acontecimientos. - parámetros se pueden pasar cuando las funciones directamente llamar a otras funciones. Usa if-else para que tus funciones se comporten

de forma diferente dependiendo de las condiciones. Felicidades si usted ha hecho a través de todo eso. Se trata de un mucho que aprender. Estamos en espera variables, cadenas, funciones, y si los demás cláusulas-, que en cierta forma son parte de toda la programación idiomas. Ahora es el momento de aprender el resto de la secuencia de comandos de sintaxis. Sólo hay un aspecto importante de script de sintaxis que aún tenemos que cubrir: bucles. Empecemos con algunas funciones más complicado, e introducir bucles.

Loops (bucles)

Algunas veces usted quiere hacer lo mismo más de una vez. Digamos, por ejemplo, que usted quería para obtener un código secreto de alguien y que quería seguir preguntando hasta que le dio el derecho código. Si sólo quería darle dos países, que podría hacer algo como esto:

var the_code = 12345;	// secret code number
var entered_number = 0;	
string entry_line[80];	// just a long empty string
function check_code()	
{	
print("please enter number..."); // display this on screen	
waitt(16);	// wait 16 ticks (1 second)
print(entry_line); // display entry string on screen	
inkey(entry_line); // requests keyboard input into the string	
entered_number = str_to_num(entry_line);	// converts input from a string to a number
if (entered_number != the_code)	
{	
print("please enter number again...");	
waitt(16);	
print(entry_line);	
inkey(entry_line);	// again, user input
entered_number = str_to_num(entry_line);	// convert entry_line to entered_number

if (entered_number != the_code)		
{		
	print("wrong again!");	
	return;	// two tries, now terminate the function
} else {		
print("wow! you've guessed it!");	// display "right" message	
return;		
}		

<code>} else {</code>		
		<code>print("wow! you've guessed it!");</code>
		<code>return;</code>
	<code>}</code>	
<code>}</code>		
<code>function main()</code>		
<code>{</code>		
<code>screen_color.blue = 128;</code>	<code>// set a dark blue background</code>	
<code>check_code();</code>	<code>// start the check_code function</code>	
<code>}</code>		

Este es un largo y algo feo función! Sin embargo, usted ha aprendido algunas nuevas instrucciones aquí. Si usted da un número entre [] corchetes en una cadena de definición, crear una cadena vacía consistente en el número de espacios. `waitt` sólo espera 1 / 16 segundos, y, a continuación, sigue la función. `inkey` produce un cursor parpadeante y permite al usuario introducir caracteres en la cadena dada hasta que pulsas [Intro]. `str_to_num` convierte un numéricos contenido de una cadena - "123", por ejemplo-para el número correspondiente. Tenemos que hacer esto porque no podemos comparar una cadena con un número en la condición de si - esto sería como comparar manzanas con naranjas. `regreso` sólo termina la función. Como puede ver, esta función no funcionará si sólo desea conservar pidiendo hasta que obtener el código número de la derecha. Y es muy feo ya imaginar si quería pedir cuatro veces en lugar de sólo dos! Te gustaría tener cuatro niveles de si-más cláusulas, que nunca es una buena cosa. La mejor manera de hacer cosas similares más de una vez es usar un bucle. En este caso, puede utilizar un bucle mantener a pedir el código secreto hasta que la persona se da por vencido. He aquí un ejemplo de un bucle en acción.

<code>function check_code()</code>			
<code>{</code>			
	<code>while (entered_number != the_code)</code>		
	<code>{</code>		
		<code>print("Please enter correct number...");</code>	<code>// display this on screen</code>
	<code>waitt(16);</code>	<code>// wait 16 ticks (1 second)</code>	
	<code>print(entry_line);</code>		
	<code>inkey(entry_line); // requests user input into a</code>		

	string		
	entered_number = str_to_num(entry_line);	// converts input from a string to a number	
	}		
	print("Wow! You've guessed it!");	// display "right" message	
}			

Se trata de un bucle while, que realmente simplifica nuestra función. Si bien los bucles vienen en esta forma general:

while (some test is true)
{
do the stuff inside the curly braces
}

Por lo tanto, las líneas anteriores decir, "Si bien la entrada no es igual al código de seguir para obtener la entrada del usuario". El bucle mantendrá la ejecución de las instrucciones dentro de las llaves hasta que la prueba es falsa. En este caso, la prueba sólo será falsa cuando los caracteres que el usuario introduce es el mismo que el número de código (es decir, "12345"). La verdadera prueba es si el número está mal - eso es lo que "!=" No! Entered_number porque se le da un valor por el str_to_num instrucción en el interior del bucle while, se no tendrá valor la primera vez que golpeó el bucle. Por lo tanto hemos definido que temprano, con una inicial valor de 0, lo que no es ciertamente el número de código. Aunque recorrer indefinidamente a menudo es útil, bucles son más comúnmente utilizados para ejecutar un conjunto de instrucciones un número específico de veces. Aquí la otro ejemplo de un bucle while que muestra cómo hacer esto. Queremos escribir un programa que primero te permite introducir un número y, a continuación, imprime el mayor número de "x" en la pantalla. Vamos a ir más de esto. En primer lugar, definir algunas nuevas cadenas y las variables:

var counter = 0;			
var entered_number = 0;			
string entry_line[80];	// a long empty string		
function count_x()			
{			
	print("please enter number..."); // display		

	"Please enter number..." on screen		
	waitt(16);	// wait a second, then continue	
	str_cpy(entry_line,"");	// clear the input line from previous input, by copying an empty string	
	print(entry_line);	// display the string	
	inkey(entry_line); // user input into the string		
	entered_number = str_to_num(entry_line);	// convert input to a number	
	str_cpy(entry_line,"");	// clear the string	
	while (counter < entered_number)	{	
		str_cat(entry_line,"x");	// add "x" to the string
		counter = counter + 1;	// increase loop
		}	
	}		
	function main()		
	{		
	screen_color.blue = 128;	// set a dark blue background	
	on_p = count_x;	// assign the count_x function to the [P] key	
	}		

On_p La declaración es similar a la on_anykey ya sabemos, pero asigna el count_x a la función [P] clave. Como probablemente has adivinado, puede asignar cualquier función a cualquier acontecimiento clave mediante el establecimiento de on_a, etc on_b a ella. Count_x dice: "mientras que la variable contador es menor que el número solicitado de la fila de 'x', añade otra 'x' a la línea y, a continuación, añadir un valor a la de la lucha". Str_cpy copiasasecond cadena en el primero, mientras que str_cat añade una segunda cadena a la final de la primera. Este bucle se mantendrá la adición de una 'x' a nuestra cadena y añadiendo 1 a el valor de luchar hasta el contador ya no es inferior a la número solicitado. El aumento de una variable de bucle es tan común que los programadores han desarrollado un método abreviado. Usando el acceso directo, mientras que el bucle se podría haber escrito como este:

while (counter < entered_number)	{	
	str_cat(entry_line,"x");	// add "x" to the string
	counter += 1;	// this was counter = counter + 1;

```
}

```

La tercera línea, `contador + = 1`, dice "añadir 1 a mí mismo". Si usted tiene `a_number = 5`, y `youwrite a_number + = 3`, es igual que escribir `a_number = a_number + 3`. You puede utilizar `--`, `*=`, y `/ =` de la misma manera. Los programadores son perezosos, que siempre están próximos a los accesos directos como este. Pruebe la función. Después de la pantalla azul aparece, pulse el botón [P]. Bueno, parece que el trabajo - sino esperar -¿qué es eso? La función sólo funciona la primera vez! Si pulse el botón [P] clave una segunda vez, hay menos 'x' muestra que hemos entrado - o ninguna' x 'a todos. ¿Qué diablos está pasando aquí?

Debugging

Si ha escrito una función, te aviso que a veces no funciona, o se comporta de manera diferente más de lo esperado. Y la frecuencia con la que aún no saben por qué. Sólo se puede pensar en dos razones: 1) A veces el lenguaje de programación simplemente no funciona. 2) Soy demasiado tonto juego de la programación. La secuencia de comandos de la sintaxis está bien probada, por lo que puede estar seguro de que 1) no es el caso. Y tú no estás demasiado mudos - bugs ocurrir a la mayoría de programadores de experimentado todo el tiempo. De hecho, es bastante normal que una función no funciona en la primera. Y es también bastante normal que, incluso una de las principales programador no encontrar la razón por sólo mirando el código.

Profesionales de depurar su código. Depurar no es gran secreto. Simplemente significa aquí el código de ejecución no como un todo, pero línea por línea, y examinar en detalle lo que está sucediendo, con el fin de encontrar posibles errores. C-Script tiene un built-in depurador. Usted puede activar por sólo añadir la siguiente instrucción en el comienzo de la función de examinar:

<code>function count_x()</code>	
<code>{</code>	
<code>breakpoint;</code>	<code>// start debugging here</code>
<code>print("please enter number...");</code>	
<code>...</code>	

Si ahora el nivel de inicio y pulse [P], el juego se congela, oímos un sonido corto y ver el siguiente línea en la pantalla:

```
<= print(@2)
```

Esta es la siguiente instrucción después de la interrupción. El @ .. es la representación interna de una directamente entró en la cadena. Ahora pulse [Espacio]. La línea se desplaza hacia arriba, y vemos algo como:

```
0.000 <= print(@2)
```

```
<= screen_txt.string = str
```

Ya hemos ejecutado una instrucción. Es la llamada de la función de impresión, por lo que son ahora dentro de esta función y ver la primera línea de la misma. Str es nuestra "Por favor, introduzca el número" cadena que es ahora asignado a screen_txt.string. Siguiendo [Espacio]:

```
15135.457 <= screen_txt.string = str
```

```
<= screen_txt.visible = on
```

El extraño número (en su PC que pueden diferir de 15135.457) es el resultado del código expresión-en este caso un puntero interno asignado a la cadena, que no debe molestarse con nosotros en la momento. Siguiendo [Espacio]:

```
16.000 <= screen_txt.visible = on
```

```
<= waitt(16)
```

Ahora hemos salido de la función de impresión y están entrando en nuestra count_x función de nuevo. Ahora véase el "por favor, introduzca el número" solicitud en la pantalla, ya que se hizo visible por la función de impresión de la última instrucción. Pulse [Espacio] algunas veces más, por lo que solo a través de la intensificación de la función línea por línea hasta llegar

```
0.000 <= inkey(entry_line)
```

```
<= entered_number = str_to_num(entry_line)
```

Ahora, con la rápida cursor parpadeante a continuación, entrar en un "2" y pulse [Intro]. El siguiente [Espacio] se

```
2.000 <= entered_number = str_to_num(entry_line)
```

```
<= str_cpy(entry_line,@4)
```

Como podemos ver, la instrucción `str_to_num` ha convertido a nuestra cadena a la número 2, como se esperaba. Para las expresiones numéricas el resultado de código simplemente muestra el número resultante. Si pulse [Espacio] dos veces más, podemos ver otro resultado significativo de código:

```
1.000 <= (counter < entered_number)
```

```
<= str_cat(entry_line,@6)
```

El resultado de un código o si al mismo tiempo la condición (o si la misma mientras no se muestra) es de 1,000 cierto, y 0,000 para falso. Por lo tanto podemos ver que la condición, mientras que aquí se cumplió, y estamos ahora en reforzar el lazo. Así, mediante la instrucción para la instrucción a través de nuestra función, estamos obteniendo una perfecta cronograma de lo que ocurre cuando una persona elige dos de la x en el símbolo del sistema: Primer paso `-entry_line = ""` (porque nos copian "" en él por `str_cpy`) `-counter = 0` (ya que lo definió con valor inicial 0) `-entered_number = 2` (porque eso es lo que ha introducido) `-0 Sea inferior a 2 para - "x" se añade a entry_line, por lo que ahora entry_line = "x" counter + = 1, por lo que ahora contador = 1 Volver en el bucle: segundo paso counter = 1 -entered_number = 2 -entry_line = "x" -1 Es inferior al 2 por lo - "x" se añade a entry_line, por lo que ahora entry_line = "xx" counter + = 1, por lo que ahora contador = 2 Volver en el bucle: tercer paso counter = 2 -entered_number = 2 -entry_line = "xx" -2 No es inferior a 2 a fin de - Caída de los bucles y estamos acabados. Ok - hasta ahora tan bueno. Ahora hemos depurado nuestra función. Por [Ctrl-Space] dejar de fumar ahora el modo de depuración y volver a la normalidad el modo de ejecución. La función se comportaron como se esperaba, pero sabemos que el problema sólo se plantea en la segunda intento. Por lo tanto, presione [P] de nuevo al paso que en el segundo tiempo. Al introducir el número, ahora entrar en "3". Ahora esperan tres bucles. Sin embargo lo que estamos recibiendo, cuando a través de la intensificación de la primero bucle es el siguiente:`

```
3.000 <= counter += 1;
```

```
<= }
```

Eso es todo - el aumento de la contra 3 ya da incluso en el primer bucle! Como consecuencia de ello, sólo una bucle y se realiza sólo una 'x' se produce. Y ahora somos capaces de encontrar la razón: La contrarrestar variable todavía tiene su valor de 2 desde la última ejecución de la función. Ahora que finalmente hemos encontró el error, es fácil corregir mediante la adición de `counter = 0;` en el comienzo de la función. No se olvide de inicializar todas las variables en la programación un función!

Utilice el depurador de la medida de lo posible. Es una gran manera de aprender cómo su trabajo y funciones cómo el lenguaje de programación que trabaja en absoluto. Por lo general los profesionales depurar sus códigos, incluso si parece funcionar - de esta manera se puede detectar "errores ocultos" que sólo se producen en determinadas circunstancias. Si estás realmente atascado con un problema en el código, encontrará en el apéndice de un colección de las más comunes las trampas de guión. Habiendo aprendido acerca de la depuración, ahora puede usted llamar a un programador junior. Para los próximos días, vamos a salir de la seca-ISH mundo de la secuencia de comandos y la sintaxis de entrar en el juego real. Aquí vamos a aprender acerca de cómo puede utilizar secuencias de comandos para definir el comportamiento del jugador, de los actores, o de la interfaz de usuario en poderosos e interesantes maneras.

Lunes: Puertas y claves

siempre es impresionante tener cosas que se mueven como parte de nuestro mundo-como Clockworks poner en marcha, la apertura de portales en secreto, o que tengan el jugador de repente encontrar agua en streaming en el toque de un botón! Por lo tanto, en este capítulo del tutorial vamos a examinar la manera de poner en marcha partes de nuestro mundo de maneras diferentes. La forma más fácil y más frecuente ejemplo de movimiento en nuestro mundo es una puerta de apertura. I Sabemos de la DMMA tutorial de cómo hacerlo: sólo tenemos que adjuntar la puerta de acción predefinida a una entidad de ruta. Entonces, si el jugador pulsa el [Espacio] clave cerca de la puerta, se gira horizontalmente en torno a su centro. Pero, ¿cómo se puede lograr esto? Echemos un vistazo más de cerca las acciones en la entidad. Entidad de Acciones

Entidades que son los objetos en movimiento de nuestro juego mundo - como las puertas, los actores, los enemigos, o el propio jugador. Su movimiento se realiza por funciones especiales - las acciones-el nombre que se les asignan. Y esas acciones consisten normalmente de una o más grandes bucles de esperar un (1) instrucción en el final.

Aquí está un ejemplo de un bucle que sólo permite girar una entidad por sí misma. Tal vez le ha surgido un ejemplo similar en la DMMA tutorial:

action entity_rotate
{
while (1){
my.pan += 3;
wait(1);
}
}

Esta es una descripción de "me rotar permanentemente con una velocidad de 3 grados por cuadro". Podemos esta acción asignar a una entidad por mie. La única diferencia entre una función y una acción que este último se define por la palabra de acción, aparece en la acción de la DMMA lista, y no tiene la paréntesis después del nombre, ya que no tiene ningún parámetro. Cada entidad de la acción se inicia una vez en el juego de inicio. Tenemos aquí simplemente a 1, mientras que la condición. 1 se considera siempre la verdad. Tan while (1) (...) Se repita las instrucciones para siempre. Esto se denomina un bucle infinito.

Dentro de este bucle, el valor 3, se añade a my.pan. Mi es un objeto como el objeto screen_txt estamos ya conocemos. Sin embargo, mi no es un objeto de texto, es una entidad objeto. Más precisamente, mi es un puntero a la entidad que la acción fue asignado. Un indicador es una referencia a algo - como una nombre temporal. Al igual que en nuestro lenguaje natural "me" o "mi" se refiere a la persona que haciendo uso de la palabra, por lo que en nuestro lenguaje de programación mi se refiere a que la entidad que ejecuta la acción. Si desea asignado este la acción no a una entidad, pero, por ejemplo, a través de una clave on_p, usted recibirá un mensaje de error ("puntero vacío") - porque mi es exclusivamente para entidades, y una clave no es una entidad! Pan es la entidad del ángulo horizontal, en grados. Este ángulo de manera permanente aumento de 3. En otras palabras: la entidad de forma rotativa. No importa si el valor excede de pan 360 grados - el motor que puede manejar. La última instrucción, espera (1), se suspende la acción para un marco de ciclo, antes de repetir el bucle. Así que tenemos una rotación de tres grados por frame ciclo. Ya hemos aprendido waitt (con dos T), que espera para el 1 / 16 segundos, mientras espera para esperar marco de los ciclos. En uno de gama baja de PC, ejecuta en sólo alrededor de 16 fps, estos dos son casi lo mismo. Ten en cuenta que todo fuera de la -función de la prestación, cambio de variable global y la ejecución de otras funciones-se llevará a cabo durante poco que esperar una pausa! Por lo tanto, cada bucle

infinito, siempre debe esperar un `waitt`. De lo contrario, el bucle nada más permitir que ocurra fuera de sí mismo, y su congelación PC (o producir un "bucle infinito" mensaje de error).

El primer movimiento

Inicio `Wed`, abrir uno de sus niveles, y crear un nuevo archivo de comandos nuevos (Mapa Propiedades -> Script -> Nuevo). Pero esta vez no vamos a hacer uso de las funciones prefabricadas. Esto

tiempo que vamos a escribir nuestro propio uno. Cierre `MIERCOLES` nuevo. Utilice un editor de texto para abra su secuencia de comandos `name.wdl` archivo, mientras que el nombre es el nombre de su mapa. Ahora pasar a la final de la secuencia de comandos de archivo, y el tipo allí su primera entidad de acción:

<code>action door1</code>			
<code>{</code>			
	<code>while (1) {</code>		
	<code>play_entsound(my,open_snd,66);</code>		
	<code>while (my.pan < 90) {</code>		
		<code>my.pan += 3*time;</code>	<code>// rotate counterclockwise</code>
		<code>wait(1);</code>	
	<code>}</code>		
	<code>waitt(16);</code>		
	<code>play_entsound(my,close_snd,66);</code>		
	<code>while (my.pan > 0) {</code>		
		<code>my.pan -= 3*time;</code>	<code>// rotate clockwise</code>
		<code>wait(1);</code>	
	<code>}</code>		
		<code>waitt(16);</code>	
	<code>}</code>		
<code>}</code>			

Ok, vamos a ver lo que hemos hecho aquí. Hemos definido un nombre de acción `door1`-una acción que puede estar conectado a una puerta de la entidad, que vamos a hacer ahora. Para la creación de una entidad de su puerta, abre una nuevo mapa llamado `mydoor.wmp`, un lugar único cubo, el tamaño es de puerta a la forma, darle una textura de madera, mover de forma que está de pie en posición vertical y el mapa origen se encuentra en su posición de bisagra y, por último, construir con la Entidad Mapa opción

seleccionada. Si no quieres hacer todo eso, sólo tiene que utilizar el `porta.wmb` prefabricados de la oficina nivel. Ahora tiene un mapa de entidad que puede colocar en alguna parte en su prueba de nivel por medio de MIÉRCOLES. Hacer ello. Luego, con la puerta en la entidad aún seleccionado del DMMA, abra la entidad en Propiedades, y haga clic en la acción `campo`. Si han hecho todo bien, ahora puede elegir su nuevo `door1` acción de la lista.

Reconstruir el nivel de utilización de Actualización de las entidades - que se basa mucho más rápido en caso de que no han cambiado nada excepto entidades. Ahora inicie Ejecutar por Nivel. Si usted no tiene un jugador en su nivel, sin embargo, puede cambiar en el movimiento directo el modo de cámara pulsando el `[0]` clave. Si lo has hecho todo bien, ahora se puede observar permanentemente una puerta de apertura y cierre, como si por arte de magia! Como ustedes recordarán, mi es el puntero a la entidad a la que la acción se adjunta - nuestra puerta. La primera mientras que la instrucción es `(1)`, lo que significa: Repetir siempre todas las instrucciones entre las siguientes par de llaves. Tenemos que repetir las instrucciones, porque queremos que esta acción a realizar, permanentemente durante el juego. De lo contrario, sólo se realiza una vez en el juego de inicio, y entonces fin. Y la puerta se puede mover un poco el paso y, a continuación, siempre de pie todavía. En el eterno loop, dejamos la puerta de jugar una apertura de sonido a través de la `play_entsound` instrucción, y continuar con otro bucle interior (se puede 'nido' como los bucles que ver). Este bucle interior aumentará el ángulo de la puerta mientras esté todavía por debajo de 90 grados. El importe para aumentar la ángulo se multiplica por el tiempo, la duración del ciclo de marco, con el fin de obtener un ángulo de incremento de la apertura de velocidad, independiente de la velocidad de cuadro.

Entonces la puerta se mantendrá todavía durante un segundo. Estamos utilizando la instrucción `waitt` aquí, que espera un cierto tiempo, a diferencia de `esperar`, que espera para un número de ciclos de marco. Después de la pausa que permite girar la puerta trasera hasta su punto de vista ha llegado a cero, y luego, tras otra pausa, todo comienza de nuevo. Estamos notando algunas deficiencias de nuestra acción. La puerta siempre gira entre 0 y 90 grados, es decir, el este y el norte, independientemente de su orientación inicial. Y se "rebase" de su final posición por parte de algunos grados, debido al hecho de que el último incremento puede añadir a un valor por encima de los 90, o por debajo de cero. Tenemos que corregir esto:

action door1			
{			
	my.skill25 = 0; // degree counter, independent of initial angle		
	while (1) {		
	play_entsound(my,open_snd,66);		
	while (my.skill25 < 90) {		
	my.pan -= 3*time;	// rotate clockwise	
	my.skill25 += 3*time;		
	wait(1);		
	}		
	my.pan += my.skill25-90;	// correct the overshoot	
	my.skill25 = 90;		
	waitt(16);		
	play_entsound(my,close_snd,66);		
	while (my.skill25 > 0) {		
		my.pan += 3*time;	// rotate counterclockwise
		my.skill25 -= 3*time;	
		wait(1);	
	}		
		my.pan += my.skill25; // correct the overshoot	
		my.skill25 = 0;	
		waitt(16);	
	}		
}			

La puerta está ahora la apertura de las agujas del reloj, y el cierre a la izquierda. La entidad habilidad, skill25, es utilizados para contar el ángulo de apertura, de modo que la puerta se abre ahora, independientemente de su valor inicial pan.

48 entidades tales variables internas que puedan ser utilizados en las acciones. ¿Por qué no podemos definir una variable var a través de una declaración para el contador? Debido a que puede tener más de una puerta en nuestro nivel. Variables que se define "el mundo" compartido por todas las acciones y funciones. Ok, la puerta se está moviendo ahora. Pero, ¿cómo podemos lograr ahora que reacciona sobre nosotros, el usuario?

Interacción con el usuario

La puerta junto a la la vista de la cámara se abre si se presiona

una tecla en el teclado. La exploración la instrucción se hace por ello. De análisis dará lugar a una entidades función caso, si está dentro de la exploración cono. Evento funciones de control de las entidades una "reacción en algo ocurre. En primer lugar vamos a escribir una acción que realiza una exploración, y que para asignar una clave:

var indicator = 0;
var my_pos[3];
var my_angle[3];
function scan_me()
{
my_pos.x = camera.x;
my_pos.y = camera.y;

my_pos.z = camera.z;	
my_angle.pan = camera.pan;	
my_angle.tilt = camera.tilt;	
temp.pan = 120;	
temp.tilt = 180;	
temp.z = 200;	// scan range – 200 quants
indicator = 1;	// this is for opening
	scan(my_pos,my_angle,temp);
}	

El conjunto de la función sólo establece los vectores que son necesarias como los tres parámetros para la exploración instrucción. Usted debe leer la descripción de exploración en el manual de referencia para comprender lo que estamos doing. We have defined dos vectores para el almacenamiento de una posición XYZ y un tres dimensional ángulo. Un vector es sólo una variable que contiene no una, sino tres números, indicado por el [3] después el nombre de la variable. Los tres números se puede acceder como. X. Y, . Z, o, alternativamente, como. Sartén, . inclinación y. rollo de los parámetros del vector. Temp predefinido es un vector que utilizamos para intermedios resultados - de aquí el ancho y el alcance de la exploración de cono. Hemos establecido el centro de la exploración de cono la posición de la cámara, su dirección a la cámara de dirección, su área de distribución a 200 quants, y su anchura a casi un semicírculo. El indicador variable se establece en 1 como una señal para la entidad cuyo caso puede obtener la función provocados por la exploración de la instrucción. Si el indicador

se establece en 1 en ese momento, la entidad sabe que esta exploración se destina para el funcionamiento de una puerta. Instrucciones de análisis podría ser utilizado para muchos otros fines también, por las explosiones por ejemplo, y tenemos que distinguir aquí para impedir que un detonante granada de repente abre todas las puertas en el nivel. ¿Cómo se puede detectar nuestra puerta de la exploración de instrucción?

define _counter skill25;	// use a meaningful name for SKILL25		
function door_event()			
{			
	if (indicator != 1) { return; } // must be right kind of scan		
	if (my._counter <= 0) {	// if door was closed, open it	
	play_entsound(my,open_snd,66);		
	while (my._counter < 90)		
	{		
	my.pan -= 3*time;	// rotate clockwise	
	my._counter += 3*time;		
	wait(1);		
	}		
	my.pan += my._counter-90; // correct the overshoot		
	my._counter = 90;		
	} else {	// otherwise close it	
		play_entsound(my,close_snd,66);	
		while (my._counter > 0) {	
		my.pan += 3*time;	// rotate counterclockwise
		my._counter -= 3*time;	
		wait(1);	
		}	
		my.pan += my._counter;	// correct the overshoot
		my._counter = 0;	
	}		
}			
action door1			
{			
my.event = door_event;			
my.enable_scan = on;	// make door sensitive for scanning		

<code>my._counter = 0;</code>	<code>// degree counter, independent of initial angle</code>		
<code>}</code>			

En primer lugar, hemos utilizado la definición de declaración para dar skill25 el nombre que describe `_counter` lo que se utiliza para. La plantilla scripts hacer uso de esta forma a entidad destrezas nombre. La puerta de acción es ahora dividida en dos, un inicio de acción y un caso de la función. La puerta se sensible para la exploración de instrucciones, y el caso está en función. Esta acción comprueba primero si se trata de la apertura de un escáner, o cualquier otra cosa. Si se trata de un derecho, es comprobar si ya se abierto o cerrado, y, a continuación, abrir o cerrar, respectivamente. Todo lo que tenemos que hacer ahora es conectar el `scan_me` función de una clave:

```
on_space = scan_me;
```

Ahora tenemos que caminar dentro de las 200 quants de la puerta y, a continuación, pulse el botón [Espacio] botón.

Preparado para más? ¿Qué acerca de un tema clave que tendrás que desbloquear la puerta? Deje que la clave de una cierta variable a 1 si tenemos que recoger, y dejar la puerta comprobar que la variable antes de la apertura. En primer lugar, tiene que definir una acción para un modelo de entidad a convertirse en un tema clave:

<code>sound key_fetch = <beamer.wav>;</code>
<code>var key1 = 0;</code>
<code>function key_pickup()</code>
<code>{</code>
<code>if (indicator != 1) { return; } // must be right kind of scan</code>
<code>key1 = 1;</code>
<code>snd_play(key_fetch,50,0);</code>
<code>ent_remove(my);</code>
<code>}</code>
<code>action key</code>
<code>{</code>
<code>my.event = key_pickup;</code>
<code>my.enable_scan = on; // pick up by pressing SPACE..</code>
<code>}</code>

Tan pronto como se acercan a la clave y pulse el botón [Espacio]

botón, que va a desaparecer, el sonido desempeñará, y la key1 variable se establece en 1. De esta manera los temas pueden ser recogidos y la influencia juego. La puerta ahora tiene que comprobar la key1 variable a la apertura:

function door_event()	
{	
	if (indicator != 1) { return; } // must be right kind of scan
if (key1 != 1) { return; }	// key must have been picked up already
...// and so on...	

Por cierto, la clave de la entidad, por supuesto, también podría ser la forma de un botón o palanca, y su cambio marco de la piel o parámetro para mostrar la activación, en lugar de desaparecer por más ent_remove. Using variables que el key1 una habilidad, podemos manejar diferentes llaves que abren puertas diferentes.

¿Y cómo podemos lograr que la puerta se abre, si ya una entidad sólo llega a cerca de que? Pedazo de pastel: O bien el jugador entidad podría realizar una exploración repetida instrucción (una vez por segundo - no más a menudo, la exploración es lento). O podríamos utilizar event_trigger en lugar de, o adicional a la exploración evento. Vamos a dejar que como un ejercicio para el lector ... hasta el martes.

Martes: Juego de Física

o darles un ejemplo por la simple física de los objetos utilizados en un juego, ahora vamos toTcreate una de las funciones más comunes: el movimiento del jugador entidad. Claro, usted no haveto ello, becauseyou puede utilizar theprefabricated player_move acción con su enorme conjunto de opciones. Pero si usted quiere llegar a ser un maestro de juego de la programación, en un punto puede que tenga que escribir su propio movimiento para funciones especiales de las entidades de su juego - y aquí youwill encountergamephysics. Itisgoing a getabitmathematical, butdon't tener miedo: los cuatro tipos básicos de aritmética será suficiente. No introducir aquí la física real. Nosotros utilizamos un forma simplificada de la física, especialmente apropiado para un juego en tiempo real donde cientos de actores realizar movimiento aritmética que han de ser lo más rápida y simple como sea posible. Al final de la día en que van a ser capaces de infundir entidades con cualquier tipo de movimiento sea el comportamiento a través de sus propias funciones. Y aún mejor, usted no tiene que esperar hasta el martes

para iniciar esta lección ... Abra su archivo de comandos nivel. Esta vez no vamos a hacer uso de los prefabricados jugador circulación funciones de la movement.wdl. En esta ocasión vamos a escribir nuestro propio uno.

action move_me
{
while (1)
{
my.x += 10 * key_force.x;
my.y += 10 * key_force.y;
move_view();
wait(1);
}
}

Las dos primeras instrucciones en el bucle while my.x y asignar nuevos valores my.y. Mi es la entidad a la que asignar la acción. My.x y my.y son las coordenadas X e Y de su posición, utilizando el mismo sistema de coordenadas que utiliza cuando la construcción del mapa por medio de MIÉRCOLES. El cambio de estas coordina los medios en movimiento, o más precisamente, teleporting la entidad. Estamos cambiando la posición mediante la adición de una expresión aritmética, la adición es realizada por el +=, y la expresión es 10 * key_force.x y 10 * key_force.y. En otras palabras, "key_force multiplicar por 10 y añadir el resultado en las entidades de la posición ". Las coordenadas X e Y de la otra variable vector key_force contienen valores numéricos que corresponde a nuestra aportación a través de las teclas del cursor. En la medida en que usted pulse el botón [arriba] clave, key_force.y adquiere el valor de 1, y siempre y cuando pulse el botón [Down] clave, el valor será -1. Así que pulsando una de las teclas del cursor, diez veces ese valor se añade en varias ocasiones, o restarse de las entidades posición, por lo tanto, la entidad se desplazan a lo largo de la X o Y de la dirección sistema de coordenadas. La siguiente instrucción se ejecutará una función predefinido-move_view-para mover el la vista de la cámara a las entidades posición. No vamos a analizar la función move_view aquí, pero sólo se agradecido de que exista - que se encuentra en el archivo de plantilla movement.wdl. La última instrucción, esperar (1), suspenda esta acción hasta el próximo ciclo del marco, cuando todas las instrucciones se repetirá de nuevo debido a la while (1). Si usted ha olvidado que esperar la instrucción, la acción nunca se interrumpido y,

por tanto, el marco del ciclo de nunca terminar. Eso sería malo.

Basta ya de la teoría. Vamos a probar lo que hemos escrito. Si MIE ya está abierto, seleccione Mapa Propiedades, y vuelva a seleccionar el mismo archivo de comandos - que tenemos que hacer que, debido a que hemos añadido una nueva entidad de acción. Coloque una entidad de un jugador del tamaño - guard.mdl por ejemplo - en el nivel, abierto Entidad de Propiedades, y haga clic en el campo de acción. Si han hecho todo bien, ahora puedes elegir su nuevo move_me acción de la lista. Reconstruir el nivel de utilización de Actualización de las entidades-que se basa mucho más rápido en caso de que no han cambiado nada excepto entidades. Ahora seleccione Ejecutar Nivel, y probar las teclas de cursor para caminar. Hmm. Bueno, debemos admitir que su primer movimiento de acción no se comporta así como la un prefabricados. Lo que pasa es que el movimiento no sigue la línea de visión del jugador -- De ello se deduce directamente de la X y Y-ejes del sistema de coordenadas. De hecho, no se puede rotar el jugador en todo. No es de extrañar, usted no ha proporcionado ninguna instrucción para eso. Y peor aún, el jugador no parar en las paredes, va a pasar directamente a través de ellos. Vamos a fijar que, en primer lugar. Con el fin de obtener la detección de colisión en su entidad movimiento, no debe cambiar la entidad coordina directamente - que telepuerto de la entidad a su nueva posición, pero el uso la instrucción para pasar un movimiento continuo a través de una distancia determinada:

<pre>var dist[3];</pre>
<pre>action move_me</pre>
<pre>{</pre>
<pre>while (1)</pre>
<pre>{</pre>
<pre>dist.x = 10 * key_force.x;</pre>
<pre>dist.y = 10 * key_force.y;</pre>
<pre>dist.z = 0;</pre>
<pre>move(my,nullvector,dist);</pre>
<pre>move_view();</pre>
<pre>wait(1);</pre>
<pre>}</pre>
<pre>}</pre>

Hemos definido un vector, dist, para almacenar la distancia a cubrir. Un vector es sólo una variable que contiene tres números, que se utilizan aquí para la X, Y, Z y coordenadas. Estamos fijando su Z coordinar a cero para evitar que el jugador

de repente volando en el aire. El paso de mi ... cambios en la instrucción el jugador posición de la misma manera como si hubiera escrito:

```
my.x += dist.x;
my.y += dist.y;
my.z += dist.z;
```

La única diferencia es que a través de movimiento, el de mi entidad - nuestro jugador - realiza una colisión la detección de más de la distancia dada por dist. Mover vector tiene dos distancias, de los cuales hemos establecido la primero un cero a usar la nullvector. La primera distancia es rotado de acuerdo a la entidades ángulos, el segundo una (utilizado aquí) no lo es. Uso de la primera rotación de distancia, en lugar de la segunda unrotated uno, nos daría la oportunidad de girar el movimiento en dirección al jugador la línea de la vista, y el uso key_force.x a cambiar su punto de vista:

```
action move_me
{
while (1)
{
my.pan -= 10 * key_force.x;
dist.x = 10 * key_force.y;
dist.y = 0;
dist.z = 0;
move(my,dist,nullvector);
move_view();
wait(1);
}
}
```

Tenga en cuenta que ahora una rotación de uso para desplazar a distancia, mientras que el unrotated distancia se establece en cero mediante el uso de la nullvector. dist.x-que otorga a la dirección antes - es ahora cambiado por key_force.y. Ahora somos capaces de hacer que el jugador rotar y mover en cada dirección. Pero todavía el movimiento parece ser desigual y antinatural. Y nuestra velocidad depende de la velocidad de cuadro - el PC más rápido la instrucción MOV se realiza con más frecuencia, por lo que el jugador cubre una mayor distancia en la misma tiempo. Que no se debe. Vamos a tratar de la figura

de una fórmula que logre buen "natural" movimientos.

La aceleración, la inercia, la fricción Supongamos que nuestro jugador se está moviendo hacia delante a una velocidad constante. La distancia que viaja aumentos, de acuerdo con su velocidad, con el tiempo: $DS = v * dt$ $v =$ velocidad en quants por garrapatas
 $DS =$ Distancia en quants $dt =$ tiempo en las garrapatas En nuestro mundo virtual-como ustedes recordarán-las distancias se han medido en quants (= alrededor de 1 pulgada) y el tiempo se mide en las garrapatas (= alrededor de 1 / 16 seg). Por lo tanto, la unidad de medida de la velocidad es quants por garrapatas. Pero, ¿qué sucede si queremos cambiar la velocidad? Tendremos que hacer una fuerza para que su impacto sobre el jugador. Cuanto mayor es la fuerza mayor es el impacto que tiene sobre la velocidad por unidad de tiempo. Sin embargo, cada cuerpo muestra una cierta resistencia en contra de dichos cambios. Esta resistencia, denominada inercia, es el resultado de la masa del cuerpo. Cuanto más grande sea la masa del cuerpo, cuanto más pequeño es el cambio de velocidad será - suponiendo una fuerza constante. Esto es lo que puede expresar en fórmulas: $dv = a * dt$ $a = F / m$ $dv =$ Cambio de velocidad $a =$ Cambio de velocidad por garrapatas (aceleración) $F =$ Fuerza $m =$ masa Hay tres tipos básicos de fuerzas vamos a tener que contar con en nuestro mundo virtual. Para empezar está la fuerza propulsora. Se trata de una variación de velocidad voluntario que, por ejemplo, puede ser inducida a través de teclado y joystick movimiento. En nuestro entorno de juego simplificado de esta fuerza deberá ser proporcional a la masa del jugador: El más grande es un hombre, más que la fuerza se puede aplicar a acelerar el mismo. $P = p * m$ La segunda fuerza que se deriva de la palabra. Puede ser una corriente que lleva el jugador en una determinada dirección, o la gravedad, tirando de él hacia abajo. La deriva de fuerzas puede variar de un lugar a otro. En nuestro mundo de juego, el importe de la deriva también es proporcional a la masa del jugador. Esto es obvio por la gravedad, pero también puede ser normal para aproximarse derivas que aplican las más fuerza a un objeto más de volumen que tiene. $D = d * m$ La tercera fuerza que tengan un efecto sobre el jugador es la fricción. Esta fuerza intenta continuamente a lo lento hacia abajo. A diferencia de la física realista, donde la desaceleración de la fuerza de fricción y de atenuación de las partes, usamos una fuerza artificial que aumenta con el jugador de la masa (la presión en el suelo) y la velocidad: $R = - f * m * v$ $R =$ fuerza de fricción $f =$ factor de fricción $v =$ velocidad $m =$ masa El coeficiente de fricción f depende de la naturaleza de la superficie, el jugador se mueve sobre el hielo con un menor coeficiente de fricción de la piedra.

Si está en suspensión en el aire, el aire de fricción es casi cero. El indicador negativo se supone que muestran que esta fuerza de la velocidad de los contadores. La masa m es también parte de la ecuación, al igual que en nuestro mundo de juego el jugador se encontrará con una mayor fricción y poner mayor énfasis en la superficie, si pesa más. Las tres fuerzas-la fuerza propulsora F , D deriva, y la desaceleración R -añadir hasta cambiar la velocidad del jugador: $dv = a * dt = (P + D + I) / m * DT = (p + d - f * v) * dt$ Así dv hay que añadir a la velocidad de cada cuadro. p , d , y f son el propulsor, la deriva y los factores de fricción aquí. dt es el tiempo durante el cual la aceleración ha cambiado la velocidad - que es el tiempo entre dos ciclos marco aquí, porque somos un nuevo cálculo de la velocidad del ciclo de cada uno de los cuadros. Como puede ver, por persuadir a todas nuestras fuerzas para ser proporcional en masa, la masa se elimina de la ecuación, al no tener influencia en el movimiento actor en la física. Podríamos traducir la última fórmula en una función correspondiente (no hay todavía la deriva):

var force[3];		
var dist[3];		
action move_me		
{		
	while (1) {	
force.pan = -10 * key_force.x;	// calculate rotation force	
my.skill14 = time*force.pan + max(1-time*0.7,0)*my.skill14;// rotation speed		
my.pan += time * my.skill14; // rotate the player		
force.x = 10 * key_force.y;	// calculate translation force	
my.skill11 = time*force.x + max(1-time*0.7,0)*my.skill11; // calculate speed		
dist.x = time * my.skill11;	// distance to cover	
dist.y = 0;		
dist.z = 0;		
	move(my,dist,nullvector);	// move the player
	move_view();	// move the camera
	wait(1);	
}		

Nuestra fuerza son 10 los factores aquí, nuestros factores de fricción de 0,7, tanto para la rotación y la traducción. Debemos utilización entidad habilidades para la velocidad, porque

tienen que ser preservados, junto con la entidad, la anterior velocidad es necesaria para nuestra fórmula. Skill114 almacena la velocidad angular y el skill111 velocidad por delante. El tiempo es el dt de la fórmula, el momento de la última imagen del ciclo. Estamos utilizando el ahora matemáticamente corregir la fórmula para el cálculo de la distancia, por lo tanto, ahora el jugador se mueve con la misma velocidad en cada PC, casi independiente de la velocidad de cuadro! Tenga en cuenta que hemos convertido nuestro original, fórmula teórica que da el cambio de velocidad por cuadro ciclo

$V \rightarrow V + DV$ $V \rightarrow V + (p - f * v) * dt$ que se expresa en la secuencia de comandos

```
my.skill11 = my.skill11 + (force.x - 0.7 * my.skill11) * time;
```

y transformado a través de los siguientes pasos

```
my.skill11 = my.skill11 + time * force.x - time * 0.7 * my.skill11;
```

```
my.skill11 = time * force.x + (1 - time * 0.7) * my.skill11;
```

finalmente, un poco más complicado

```
my.skill11 = time * force.x + max(1 - time * 0.7, 0) * my.skill11;
```

¿Por qué esto, y lo que para el máximo? Max (a, b) ofrece el mayor número de uno o b. Through max (1 -- * tiempo 0.7, 0) estamos limitando el resultado a valores positivos. Tenemos que hacerlo, porque de lo contrario muy baja en las tasas de marco - y el momento de los valores de tiempo * 0.7 puede ser mayor que 1, dando un resultado negativo. Esto invertir la velocidad - el jugador se mueva hacia atrás! Esa sutil diferencias entre el ordenador y simular la verdadera realidad siempre debe ser considerado.

(Normalmente, incluso un programador con experiencia de juego no aviso tal efecto antes de un acción se comporta de manera diferente de lo esperado). Ahora vamos mucho más fluida a través de los pasillos gracias a la nueva acción de mover. Nosotros son capaces de acelerar suavemente y ralentizar la misma manera. Pero, ¿qué ocurre si el jugador encuentra una escalera, o un abismo?

Caída

Si bien nos estamos moviendo horizontalmente nuestro reproductor pulsando las teclas, su movimiento vertical, se causados por su medio ambiente. Si se encuentra en terreno sólido, no debe realizar ningún vertical movimiento en absoluto. Pero si es de repente flotando en el aire - este puede ser el caso si fuera a caminar sobre el borde de un abismo - debe caer. Para ello tenemos que determinar su altura sobre el el terreno. Estamos utilizando la traza de instrucciones para ello.

action move_me	
{	
while (1) {	
force.pan = -10 * key_force.x;	// calculate rotation force
my.skill14 = time*force.pan + max(1-time*0.7,0)*my.skill14;// calculate rotation speed	
my.pan += time * my.skill14; // rotate the player	
	my.skill11 = time*force.x + max(1-time*0.7,0)*my.skill11; // calculate speed ahead
	dist.x = time * my.skill11; // distance ahead
	dist.y = 0;
	vec_set(temp,my.x);
	temp.z -= 4000; // calculate a position 4000 quants below the player
	// set a trace mode for using the player's hull, and detecting map entities and level surfaces only
	trace_mode = ignore_me+ignore_sprites+ignore_models+use_box;
	dist.z = -trace(my.x,temp); // subtract vertical distance to ground
	move(my,dist,nullvector); // move the player
	move_view(); // move the camera
	wait(1);
}	
}	

Trace devuelve la distancia a la siguiente obstáculo a lo largo de un rayo entre dos posiciones. Estamos utilizando el jugador centro (my.x) en la primera posición, y se ajuste la temperatura de vectores a una segunda posición algunos miles de quants a continuación. A través de la use_box tracemodewe están dando thetrace Raya "espesor" del jugador del modelo de casco - que normalmente tiene 32 quants diámetro - y también el retorno la distancia no a la del jugador centro, pero a su caja. Eso es

los pies en este caso, porque que son la localización verticalmente hacia abajo. Por lo tanto, rastrear calcula la distancia entre los pies del jugador y el primer mapa o por debajo de la superficie del terreno. Si sus pies están por debajo del nivel del suelo, lógicamente traza devuelve un valor negativo. Hemos tomado el camino más fácil, simplemente moviendo el jugador verticalmente por exactamente la cantidad de distancia entre sus pies y el terreno. Dist.z Así se establece en la distancia vertical a cubrir. Ahora estamos en condiciones de caminar las diferentes regiones de la superficie con alturas de nuestro jugador, al igual que las escaleras. Por favor Tenga en cuenta que para la sencillez que utiliza la distancia relativa a moverlo en la dirección Z. Normalmente, la distancia absoluta que se ha utilizado aquí, pero siempre y cuando el jugador no se inclina, no importa. Lamentablemente, el jugador ahora se adapta a las diferentes alturas como unnaturally y desigual como con nuestros los intentos iniciales en adelante los movimientos. Para lograr más suave y más natural que los movimientos de nuevo tendrá que echar un vistazo a las fuerzas que tienen un impacto sobre el jugador de los movimientos verticales. -En la medida en que el jugador está en suspensión en el aire -i.e. trace devuelve un valor por encima de 0-sólo la gravedad y la fricción del aire tienen un impacto en él. Gravitación es una deriva que ya hemos encontrado: da el jugador a la baja aceleración ... -... hasta que el jugador toca tierra o se hunde en él-es decir, trace devuelve un valor inferior o igual a 0. En este caso, una fuerza resistente adicional surta efecto. Este es un nuevo tipo de fuerza que se más fuerte es el más profundo que el jugador se hunde en el terreno. Se le induce al alza con un aceleración. Además, la fricción aumenta significativamente cuando el terreno es penetrado. Dependiendo del jugador entidades centro, el jugador de los pies son capaces de penetrar incluso las superficies de duro, impenetrable roca! En el mundo real este tipo de superficie, por supuesto, nunca dar paso, pero la rodilla del jugador que las articulaciones, que produce el mismo efecto. Así pues, la resistente fuerza de resultados de la elasticidad de sus articulaciones o-si el jugador es su motor, la suspensión del vehículo. Igualmente nos puede explicar el aumento de la fricción de tierra por la fricción de sus músculos o del choque del vehículo absorbedores.

var friction;	// calculate rotation force		
---------------	-----------------------------	--	--

action move_me			
{			
while (1) {			
force.pan = -10 * key_force.x;			
my.skill14 = time*force.pan + max(1-time*0.7,0)*my.skill 14;// rotation speed			
my.pan += time * my.skill14; // rotate the player			
vec_set(temp,my.x);			
temp.z -= 4000;	// calculate a position 4000 quants below the player		
// set a trace mode for using the player's hull, and detecting map entities and level surfaces only			
trace_mode = ignore_me+ignore_sprites +ignore_models+use_box;			
result = trace(my.x,temp);// subtract vertical distance to ground			
		if (result > 5) {	// in the air?
		force.x = 0;	// no pushing force
		force.y = 0;	
		force.z = -5;	// gravity force
		friction = 0.1;	// air friction
		} else {	// on or near ground
		force.x = 10 * key_force.y;	// translation force
		force.y = 0;	
		force.z = -0.5 * result;	// ground elasticity
		friction = 0.7;	// ground friction
		}	
		my.skill11 = time*force.x + max(1-time*friction,0)*my	

		.skill11; // calculate ahead speed	
		my.skill13 = time*force.z + max(1-time*friction,0)*my.skill13; // calculate vertical speed	
		dist.x = time * my.skill11;	// distance ahead
		dist.y = 0;	
		dist.z = time * my.skill13;	// distance down
		move(my,dist,nullvector); // move the player	
		move_view();	// move the camera
		wait(1);	
	}		
}			

A través de la instrucción, si somos capaces de distinguir si el jugador está en suspensión en el aire o dentro de los 5 quants de la tierra. En el primer caso no será impulsado por las teclas del cursor, pero de forma por una constante fuerza de gravedad. En este último caso una fuerza de resistencia, siendo proporcional a la (negativa) depthof inmersión (resultado de la traza), trata de empujar lo de la tierra. Estamos utilizando un entidad más habilidad, skill13, para el almacenamiento de la velocidad vertical. Bueno ... ahora podemos añadir más geniales características de nuestra acción. Podríamos usar más claves para dejar que él mirar hacia arriba y hacia abajo, o saltar, o de pato. Nuestro script que crecer más y más grandes, hacia el tamaño de prefabricados player_move la acción en el movement.wdl. Ahora vamos a dejar que a usted. En menos hasta el miércoles.

Miércoles: la inteligencia artificial

la cosa más difícil en juego el desarrollo de juegos es cómo definir el comportamiento inteligente a los opositores. Y, por lo tanto, antes de que realmente llegado a su fin con este tutorial, vamos a ocupado T nosotros mismos un poco con la creación de vida artificial inteligente. En orden a las personas de nuestro mundo con las criaturas vivientes, que normalmente hacen uso de un modelo de entidades. Modelo entidades son los objetos que son capaces de animar, y normalmente se utilizan para los actores, los opositores, o monstruos. Monstruos que quieren ser tomadas en serio tienen que comportarse en un "realista" manera. Tienen que reaccionar a los jugadores inteligentes en

los medios, eludiendo al mismo tiempo que él sigue siendo fuerte y cazar en él cuando muestra signos de debilidad. Una especie de inteligencia artificial que se necesita para ello. En este capítulo del tutorial usted aprenderá a infundir sus criaturas con una personalidad electrónica. Vamos a hacer de ello una excursión a la teoría de máquinas de estado. En la clausura de el capítulo que será capaz de crear seres electrónicos que, en su complejidad son comparables a simple organismos.

La teoría de las cajas de negro El término Inteligencia Artificial es sinónimo a los métodos que dan las máquinas la capacidad de hacer decisiones sensatas. En lo que respecta a nuestros propósitos son que se trata, es de ninguna importancia si la máquina en realidad no poseen inteligencia. El objetivo es hacer que se comportan como inteligente como sea posible. Si es realmente posible hacer declaraciones en cuanto a una máquina de inteligencia sobre la base de que la el comportamiento ha sido en gran medida una cuestión de debate entre Behaviourists, Mechanists, dualistas, y los discípulos de otras teorías desde hace muchos años. Pero esto, sin duda, muy interesante discusión no tiene la más mínima un poco de influencia en secuencias de comandos. Como la estructura interna de la máquina no es de interés para nosotros por el momento, vamos a considerar es un cuadro negro que se define únicamente por su comportamiento observable, que las acciones y reacciones. Si el comportamiento de nuestra máquina se caracteriza por una cierta simplicidad puede ser descrito como un estado máquina. El comportamiento de una máquina de estados que pueda considerar un número determinado de manera clara pautas de comportamiento independiente. Cada uno de estos patrones de comportamiento es igual a un interior state. Thusa máquina de estados tiene un número limitado de estados disponibles, es siempre 'en' uno de estos estados. El Estados de una manera constituyen la "vida interior" de la caja de negro. Para cada estado existen circunstancias, por ejemplo, un estímulo externo, que causa la máquina para cambiar en un estado diferente. Por ejemplo, un actor en un shooter en primera persona podría tener los siguientes estados de comportamiento: Esperando, Atacar, la CESPAP (e) Ing, D (es decir) Ying, y muertos.

Las flechas indican las transiciones entre los estados. Estas transiciones pueden ser desencadenadas por los acontecimientos en las secuencias de comandos. Aquí está una lista de

acontecimientos que pueden causar un actor para responder por el cambio entre estados:

Event	Realised by
Actor spots Player.	Function with periodic trace instruction and evaluating the result. Be careful, traceis slow!
Player comes close to Actor (or vice versa).	event_trigger
Player and Actor touch.	event_entity, event_impact
Player has removed himself from the Actor beyond a certain distance.	Function that periodically calculates the distance to the player entity.
Actor is near an exploding object.	event_scan
Player hits Actor with the trace instruction.	event_shoot
An animation cycle of the Actor model has passed.	Function that periodically compares the frame parameter.
A variable that is being changed by another function got a certain value.	Function that periodically compares that variable.
A certain amount of time has passed.	Function which performs a waitt,or counts down an entity skill.
Random event.	Function that periodically compares an expression containing random().

Cada uno de estos eventos pueden, por supuesto, ser combinado con cualquier otro. Cualquiera de las entidades principales de su acción o evento función es responsable de la transición entre estados. Aparte de los estados que también son capaces de definir variables interior de la máquina que puede adquirir diferentes valores e influir en el comportamiento. Nuestra entidad habilidades interiores son variables. Sus valores pueden ser realizados en el uso de una función y, por ejemplo, decidir sobre la transición a otro estado.

El cuadro de los Estados

La división del comportamiento de nuestra máquina en los estados y transiciones nos permite mantener la actor la acción de un poco transparente. Para dar un ejemplo, ahora vamos a definir nuestro primer estado máquina actor. Para los efectos de la presente que vamos a hacer uso de uno de los robots a partir de la juego de ordenador "Misión". Antes de escribir la primera línea de la secuencia de comandos, vamos a crear un concepto teórico. ¿Qué tipo de Estados nuestro robot es que, ¿cómo son estos estados va a ser distinguidos unos de otros, lo que variables y hacer las transiciones lo que necesitamos? Al comienzo del juego, el robot se Espera en el estado. Esto significa que se esconde en algún rincón oscuro de espera para el jugador. Si el jugador llega cerca de él va a cambiar en el ataque estado. Si el jugador lo golpea con su arma, el siguiente

paso depende de la del robot resto de la salud, es decir, el número de visitas que ya había sido obligado a tomar: el próximo estado ya sea de ataque, Escape, o morir.

State	Player near	Health > 30	Health <= 30	Health <= 0	Dying finished
WAIT	ATTACK	ATTACK	ESCAPE	DIE	DEAD

Para mantenerse al tanto de la salud vamos a emplear uno de los robot de la entidad habilidades. Para una mejor transparencia que resumen los estados en una tabla de estados. Hemos definido cinco eventos para el volante que pueden cambiar su estado. Tres de ellos son en función de una salud interior habilidad. Nosotros podemos añadir ahora la resto de los estados a la mesa.

State	Player near	Health > 30	Health <= 30	Health <= 0	Dying finished
WAIT	ATTACK	ATTACK	ESCAPE	DIE	DEAD
ATTACK	ATTACK	ESCAPE	DIE	DEAD	DEAD
ESCAPE	ESCAPE	ESCAPE	DIE	DEAD	DEAD
DIE	DEAD	DEAD	DEAD	DEAD	DEAD

Nuestro robot es una máquina bastante simple: si un comportamiento más complejo se requiere que nuestra mesa contienen mucho más columnas y líneas. En la inspección más cerca usted podría notar que unos pocos transiciones innecesarias se definen en la tabla. Por ejemplo, la transición entre Espera y Muerto nunca se hizo porque el robot siempre transiciones a Die primero. Sin embargo, dado que con más máquinas de estado complejas casos especiales como este no siempre son tan fáciles de ver más es siempre sensible a definir el cuadro tan completamente como sea posible. Una vez que el cuadro de los estados ha terminado la programación se inicia. Podríamos escribir todo junto en una gran acción, o definir diferentes funciones para cada estado. Elegimos el segundo enfoque - que ofrece la ventaja de que las funciones del Estado pueden ser reutilizados para diferentes máquinas.

action my_robot		
{		
	my._walkframes = 1;	
	my._entforce = 0.7;	
	my._armor = 100;	

	my._health = 100;	
	my.enable_scan = on;	// sensible of explosions
	my.enable_shoot = on;	// sensible of gunshots
	my.event = myfight_event; // handle hits	
	mystate_wait();	// First state: watch for the player
}		

De ahora en adelante haremos uso de la entidad funciones predefinidas y variables en movement.wdl y actors.wdl. En un primer momento nos están dando el número de fotogramas para el caminar, correr, atacar y morir que los ciclos de nuestro robot, enemy1.mdl, se compone de. Todos los parámetros a partir de un guión bajo (_) son habilidades entidad, que se dan los nombres por los senseful definir las palabras clave en el movement.wdl.

Ahora tenemos que definir el caso de la función, que tiene que manejar hits, y el mystate_wait, mystate_attack, mystate_escape, y mystate_die funciones, que representan las diferentes los estados de la máquina. Para los Muertos estado que no necesitamos una función: un enemigo muerto no se supone para avanzar más.

Empecemos con la función de morir. Es la más simple una:

function mystate_die()
{
my._animdist = 0; // reset entity's animation time to zero

	while (my._animdist < 100)
	{
	ent_frame("death",my._animdist); // set the frame from the percentage
	my._animdist += 5 * time; // calculate next percentage for death in ~1.25 seconds
	wait(1);
	}
	my.enable_scan = off; // become insensible
	my.enable_shoot = off;
	my.event = null; // and don't react anymore
}	

Esto es sólo animación. my._animdist da el porcentaje de la muerte de animación de la modelo. Es la primera serie a 0, entonces definitivamente el aumento de la animación en bucle hasta que haya alcanzado el 100%. El ent_frame instrucción alterna a través de la animación de fotogramas que comienzan con el nombre de "la muerte". Para prueba de la acción, vamos a añadir una simple

mystate_wait y función de un evento que no hace nada, pero la recepción de resultados:

function myfight_event()
{
if (((event_type == event_scan) && (indicator == _explode)))
((event_type == event_shoot) && (indicator == _gunfire))
{
if (my._armor <= 0) { my._health -= damage; }
else { my._armor -= damage; }
}
}
function mystate_wait()
{
while (1) {
if (my._health <= 0) { mystate_die(); return; }
wait(1);
}
}

Tenga cuidado al escribir las funciones my_robot antes de la acción, porque este último necesita el primero. El evento contará función de la habilidad armadura entidad, ya sea por si una explosión, un disparo o ha nuestro éxito actor. El indicador de las variables y los daños son fijados por la correspondiente función de armas. Como usted ha aprendido el día de ayer, el indicador se utiliza para distinguir la exploración o disparar caso de otros exploraciones que se utilizan no para explosiones, pero para otros fines. Y representa el daño la fuerza de las armas. Si no hay ningún resto de armaduras, la salud se reducirá en más Hits mystate_wait hasta que la función inicia morir si la salud ha llegado a cero. Ahora ponga el robot y un arma en su nivel. Usted puede construir una nueva arma, o utilizar un sparkgun o un flashgun de weapons.wdl. Asignar a la my_robot actor, caminar en el nivel, agarrar el arma, y comenzar a fuego en el indefenso robot. Si has hecho todo bien, después de unos 10 sparkgun éxitos que se muerda el polvo. Bueno, no es justo para disparar un actor que no puede disparar hacia atrás. Por que tiene que detectar en primer lugar. Usaremos la instrucción de exploración para ver. El jugador se ha fijado enable_scan si la acción es player_move utilizado, por lo que el robot puede utilizar event_detect para ver si se ha escaneado el jugador.

function myfight_event()	
{	
	if (((event_type == event_scan) && (indicator == _explode))
	((event_type == event_shoot) && (indicator == _gunfire)))
{	
	my._signal = 1; // by shooting, player also gives away his position
	if (my._armor <= 0) { my._health -= damage; }
	else { my._armor -= damage; }
}	

	if ((event_type == event_detect) && (you == player)) {
	my._signal = 1; }
}	
function mystate_wait()	
{	
while (1) {	
if (my._health <= 0) { mystate_die(); return; }	
if (my._health < 30) { mystate_escape(); return; }	
	// scan for the player
	temp.pan = 180;
	temp.tilt = 180;
	temp.z = 1000;
	indicator = _watch;
	scan(my.x,my.pan,temp);
	if (my._signal == 1) { mystate_attack(); return; }
	waitt(8);
	}
}	
action my_robot	
{	
my._walkframes = 1;	
my._entforce = 0.7;	
my._armor = 100;	
my._health = 100;	
my._signal = 0;	// player not yet detected
my.enable_scan = on;	// sensible of explosions
	my.enable_shoot = on; // sensible of gunshots
	my.enable_detect = on; // sensible of the player
	my.event = myfight_event; // handle hits & detection
	mystate_wait(); // first state: watch for the player
}	

En este sentido, hacer uso de una entidad habilidad que hemos

llamado `my._signal` (`_signal` se define en la `movement.wdl`) para permitir que diferentes acciones de la entidad simple intercambio de mensajes - en este caso que la entidad ha detectado el jugador. La función de escaneo `mystate_wait` para el jugador en 1000 quants delante del robot. Tenemos que establecer la variable indicador para asegurarse de que nuestros la exploración no se confunde con una explosión o la apertura de una puerta. Al cambiar de espera (1) a `waitt` (8) hemos reducido la tasa de bucle en el `mystate_wait` función para escanear sólo dos instrucciones por segundo. De análisis es lento, y simultánea de exploraciones cientos de robots, de otro modo, reducir la velocidad de cuadros. El robot se necesita medio segundo ahora a detectar el jugador. Este es su tiempo de reacción. Tenga en cuenta que el uso de la exploración al jugador se le detectó a través de las paredes. Para evitar esto, un rastro la instrucción debe seguir para comprobar si el jugador es realmente visible (hemos omitido aquí que, pero lo encontrará en el `WAR.WDL`). Cada entidad ha encontrado el conjunto `enable_scan` dará una detectar caso, pero sólo el jugador establecerá `my._signal` a 1, que es la señal para atacar:

<code>function mystate_attack()</code>
<code>{</code>
<code>my._animdist = 0;</code>
<code>while (1) {</code>
<code>if (my._health <= 0) { mystate_die(); return; }</code>
<code>if (my._health < 30) { mystate_escape(); return; }</code>
<code>// turn towards player</code>
<code>temp.x = player.x - my.x;</code>

<code>temp.y = player.y - my.y;</code>	
<code>temp.z = player.z - my.z;</code>	
<code>vec_to_angle(my_angle,temp);</code>	
<code>force = 4;</code>	<code>// set rotation speed</code>
<code>actor_turn();</code>	<code>// rotate towards my_angle</code>
<code>ent_cycle("attack", my._animdist);</code>	
<code>my._animdist += 5 * time;</code>	
<code>if (my._animdist > 100) {</code>	
<code>my._animdist -= 100;</code>	
<code>// fire at player at end of animation cycle</code>	
<code>shot_speed.x = 100;</code>	
<code>shot_speed.y = 0;</code>	
<code>shot_speed.z = 0;</code>	
<code>my_angle.pan = my.pan; // tilt is already set from</code>	
<code>vec_to_angle</code>	
<code>vec_rotate(shot_speed,my_angle);</code>	

<code>// now shot_speed points ahead, towards the player</code>	
<code>damage = 20;</code>	
<code>fire_mode = 303.2; // orange fireball, with smoke</code>	
<code>create(fireball,my.x,bullet_shot);</code>	
<code>}</code>	
<code>wait(1);</code>	
<code>}</code>	
<code>}</code>	

Esta función contiene unos 3-D cálculos. Cuando el robot ha detectado al jugador, que tiene que ir a su vez hacia él, y comenzar a tirar fireballs. En primer lugar, la dirección del robot para el jugador es calculado. Estamos utilizando la variable de tiempo vectoriales a la dirección - que es a la distancia entre dos puntos en el espacio. Como puede ver, un vector de dirección se puede calcular simplemente por restando los tres componentes del vector de la posición inicial de la posición de destino. Nosotros no explicar vector aritmética aquí - si no me crees, eres libre de comprar un libro de álgebra lineal. El siguiente paso es convertir esa dirección en los ángulos, utilizando el `vec_to_angle` instrucción. El `pan` valor de la `my_angle` vector se convierte en el ángulo horizontal entre el robot y jugador, y el `tilt` valor de la inclinación se convierte en el ángulo vertical entre ellos. La función predefinida `actor_turn` (definido en `actors.wdl`) gira una entidad horizontalmente hacia un objetivo con un ángulo determinado la velocidad por la fuerza variable. Se podría establecer directamente nuestro robot del `pan` a `my_angle.pan` ángulo, pero entonces `turn` unnaturally que a su vez rápido, y por lo tanto, siempre se enfrentan al jugador. Siguiente toca atacar a la animación. Al llegar el último cuadro, que tiene que tirar una bola de fuego. Nosotros está utilizando la función `bullet_shot` aquí. Esta función fue destinado a nuestra flashgun de armas, pero es un buen ejemplo de cómo puede volver a utilizar casi cada función para diferentes propósitos. `bullet_shot` requiere la `shot_speed` vector que se fijará a la derecha la velocidad y dirección. Nosotros haremos que al establecer que a 100 quants a la dirección X, entonces la rotación en la dirección horizontal nuestro robot se enfrenta, mientras que la reutilización de la `my_angle.tilt` calculado previamente el valor de dejar la bola de fuego dirección vertical hacia el punto al jugador. `vec_rotate` hace el trabajo. En este punto, una vez más nuestro rompecabezas para el lector: Las dos primeras líneas en el bucle de ataque, que tanto si comienzan con `my._health ...`, se debe dar exactamente en ese orden. ¿Por qué?

El último estado restantes, *Escape*, es pedazo de pastel ahora:

```
function mystate_escape()
{
while (1) {
if (my._health <= 0) { mystate_die(); return; }
// turn away from player
temp.x = my.x - player.x;
temp.y = my.y - player.y;
temp.z = my.z - player.z;

vec_to_angle(my_angle,temp);
force = 4;
actor_turn();
// walk away
actor_move();
wait(1);
}
}
```

La única función nueva que estamos pidiendo aquí es `actor_move`, que se define en la `movement.wdlscript` y permite al actor a pie por delante.

Avanzado estado de máquinas

Inteligente y "personalizado" el comportamiento de los opositores da una mayor satisfacción en particular con juegos de rol o juegos de acción. Una vez que han venido desempeñando un juego para un determinado periodo de tiempo que se iniciará

Aviso a ciertas características de sus oponentes-los puntos débiles o idiosincrasia-que puede hacer uso de. Usted es capaz de desarrollar una estrategia. Si estas características son insuficientes, por lo que es una estrategia: el juego será una mera primitiva disparar - «em-hasta de juego. Tales juegos rápidamente convertirse en aburrido. ¿Cómo podemos mejorar nuestro robot? Más evidentes son las posibilidades de estado ampliaciones, que dar al actor una más compleja, más "personal" repertorio de comportamiento. Él podría dar un grito de cólera cuando ve el jugador, que podría golpear al rebote, o empezar a girar. El ataque, que podría avanzar hacia el jugador, que esquivar hacia los lados, a fin de evitar un impacto directo. Algunos de estos posibilidades han sido realizadas en el `war.wdl` archivo. Mejoras del actor estratégico del comportamiento son mucho más complicados de realizar, y no a menudo se encuentran en los juegos de acción. Que claramente moverse en la dirección

del desarrollo de un mayor grado de la inteligencia artificial y sólo pueden ser realizadas por dejar que los actores se comunican el uno con el otro (exploración a través de instrucciones y variables) y / o haciendo que se comporten en lo que respecta a la la posición del jugador en relación con el nivel de la topografía. Algunas sugerencias más para que así:
-Tan pronto como un actor («guardián») considera que el jugador que las alarmas un colega, el actor dentro de un determinado distancia. Adecuado para plantear una alarma de los objetos son como sensores, cámaras, cables de viaje-, la luz o las barreras.

Es importante para la construcción de suspense que el jugador es consciente de la alarma! -Si varios agentes ataque el jugador, que tratan de rodear y rodear él. -Los actores se encuentran en espera para el jugador, que acechan alrededor de las esquinas, por avanzar hacia el jugador sólo hasta el punto de que él aún no puede ver. -Puede haber cañones y habitaciones en el nivel, que funcionan como 'trampas': aquí, los actores pueden ataque el jugador de todas las partes. Los actores tratan de molestar al jugador o le empuje en estos casos. Actores-puede ofrecer un pronóstico sobre el resultado de la lucha por la comparación de la del jugador fuerza a los suyos. Si el jugador es todavía muy fuerte, que ellos llaman re-aplicación, si no, que ataque directamente. Si ven que el jugador atacado ya, ingresen en el ataque para combinar fuerzas. Sin embargo, nunca debe exagerar esto. No es de ajedrez. El jugador siempre debe ser consciente de que hay peligro por delante. Actores que ocultar por completo y se encuentran en emboscada para el jugador durante bastante tiempo en de repente para llegar a él desde todas las partes para terminar fuera de él no son mucho más divertido tiene en su juego.

Jueves: La interfaz de usuario

hasta ahora todo lo que se preocupa exclusivamente la construcción de un mundo virtual y de sus habitantes. En este capítulo vamos a hablar de las funciones que, si bien se extiende fuera del mundo, todavía son parte del juego. Estas son las funciones de interfaz de usuario que hacen posible la comunicación entre el usuario, sentado cómodamente delante de la pantalla de su ordenador, y su homólogo imaginario al jugador, que es la entidad que tiene que ejecutar a través de los laberintos subterráneos y hacer el trabajo húmedo. U Textos Si el jugador tiene algo que decirle al usuario, esto puede hacerse en el simple texto escrito. Textos se utilizan para la presentación de informes en la pantalla, sino que también puede ser utilizado para

un diálogo entre el jugador y los actores en un juego de aventuras. Al igual que ocurre con mapas de bits, texturas, objetos y hay una "jerarquía" de los textos, que van desde el crudo sin texto a la jazzed de seguridad en la pantalla informe. Como recordamos, 'crudo' texto se define como una secuencia de caracteres con la palabra clave de cadena:

```
string my_string = "this is a text!!";
```

La cadena incluso puede constar de varias líneas, ya sea que correspondan a varias líneas en el archivo de comandos o están separados unos de otros, por el "\ n" cadena de caracteres

```
string my_string_3 =  
"this is the first line,  
this is the second,\nand this, finally, is the third line."
```

El "\ n" entre la segunda y tercera línea, por supuesto, no aparecen en la pantalla. Con el fin de el texto a ser visible en la pantalla, en primer lugar, tenemos que definir un conjunto de caracteres (font) primero. Una fuente está simplemente compuesto de imágenes poco para cada letra. Las imágenes aparecen en un cierto orden dentro de un mapa de bits. Esta orden corresponde a los primeros 128 o 256 caracteres de la PC el conjunto de caracteres (ASCII agrupación).

Todos en todos los mapas de bits debe dejar espacio para 128 o 256 letras, números y símbolos, cada personaje teniendo exactamente la misma cantidad de espacio. La primera fila permanece vacío, la tercera fila contiene el uso de las mayúsculas, la cuarta contiene las letras pequeñas. El espacio entre los caracteres se puede ajustar a la transparencia del color 0. Ni que decir tiene que todos los imaginables conjunto de caracteres-por ejemplo, Símbolos griegos, personajes extraterrestres o los derivados de su propia imaginación-puede ser representados. En nuestro ejemplo, el alemán 'diéresis' en el lugar de los caracteres especiales '[' , '|', ']' y '(', '\', ')'. En la plantilla de un simple directorio de mapa de bits (ackfont.pcx) se añadió que pueden servir como un ejemplo para la agrupación de su propio conjunto de caracteres.

La palabra clave de fuente se utiliza para definir el conjunto de caracteres tales como:

```
font standard_font = <panfont.pcx>,8,10;
```

Los dos últimos parámetros a la anchura y altura en píxeles de un solo carácter dentro de la de mapa de bits conjunto de caracteres (incluyendo los vacíos). Estas 128-un carácter de mapa de bits tiene que incluir exactamente $8 * 10 * 128 = 10240$ píxeles. Fuente mapas de bits de 11, 128, o 256 caracteres son posibles. Una vez que tenemos la fuente disponible que podemos hacer un texto fuera de nuestra cadena:

```
text my_text {  
pos_x = 20;  
pos_y = 40;  
font = standard_font;  
string = my_string;  
}
```

Pos_x y pos_y regreso a la posición del texto en la pantalla en píxeles, en referencia a la parte superior izquierda del esquina. Pero tendremos que hacerlo visible en primer lugar, porque el texto se ha definido anteriormente todavía no se mostrará en la pantalla. La siguiente función hace que nuestro texto aparece en la pantalla:

```
function show_my_text() { my_text.visible = on; }
```

Este método nos permite mostrar varios textos en pantalla simultáneamente. Como es posible para nosotros cambiar el texto de las posiciones de pos_x y pos_y durante el juego, el texto también puede hacerse a rodar a través de la pantalla verticalmente:

function roll_my_text()		
{		
my_text.pos_y = 480;	// move the text below the screen	
my_text.visible = on;	// make it visible	
	while (my_text.pos_y > 0) {	// as long as text still is on-screen
	my_text.pos_y -= 1; // roll upwards one line of pixels	
	wait(1);	// then wait for one frame
	}	
	waitt(16);	// time enough to read the last line

	<code>my_text.visible = off; // shut off the text</code>	
	<code>}</code>	
	<code>on_p = roll_my_text;</code>	

La función se inicia pulsando el [P] clave. Hemos construido un bucle while en este; el texto el movimiento y esperar instrucciones se repiten en cada ciclo de marco. Sólo si el valor de pos_y después mucho restando de 1 ha llegado finalmente a 0, es decir, si se toca el extremo superior de la pantalla, mientras que la bucle no se ejecutará más. La función permite a continuación, el texto desaparece, y termina.

Paneles

Tenemos muchos juegos utilizan pantallas que muestran los números de un bar o dar información sobre el estado del jugador o algunas variables del juego. Un ejemplo clásico sería el papel que juega el juego la salud, luchar contra la fuerza, y docenas de otras características-jugador en constante cambio y tienen influencia en el juego. Un panel en la pantalla representa dicha muestra. Para dar un ejemplo de un grupo que ahora representan los valores de algunas variables en la pantalla en forma de números. Es muy útil disponer de las variables siempre presente antes de los ojos, mientras que las pruebas-depuración-afuncion usted tiene escrito usted mismo. Paneles se definen de manera similar a los textos. En lugar de una cadena de este tiempo, hay definiciones para numérico muestra:

<code>font standard_font = <panfont.pcx>,8,10;</code>
<code>panel debug_panel {</code>

	<code>pos_x = 2;</code>
	<code>pos_y = 2;</code>
	<code>flags = visible,refresh; // make it visible and displayable over the view</code>
	<code>digits 0,0,3,standard_font,16,time_fac;// frames per second</code>
	<code>digits 40,0,3,standard_font,1,camera.pan;</code>
	<code>digits 80,0,3,standard_font,1,camera.tilt;</code>
	<code>digits 120,0,4,standard_font,1,camera.x;</code>
	<code>digits 160,0,4,standard_font,1,camera.y;</code>
	<code>}</code>

Ahora tenemos algunos numérico aparece en el extremo superior de

la pantalla que nos informará acerca de la del juego o el estado del jugador. Cada línea de dígitos en el panel de definición muestra una variable numérica o parámetro. 8 dígitos contienen parámetros, que determinan la X y Y-posición de la pantalla en el panel, el número de dígitos, la fuente, un factor de multiplicación, y la variable o parámetro en sí. La posición de valores a la distancia en píxeles de la esquina superior izquierda de la pantalla numérica de la esquina superior izquierda del panel. El pabellón de actualización hace que el panel que se reconstruida constantemente. Esta es la única manera de que se muestra por encima de la vista de cámara, que, como así se genera con cada nuevo cuadro. Con textos no tenemos que hablar de la bandera de actualización porque no se ajusta de forma automática. En el panel ahora podemos ver-de izquierda a derecha-el marco actual tasa de fotogramas por segundo, y cuatro más que los parámetros de la cámara ver la posición. Si quisiéramos seguir el estado de otras variables durante el juego que puede mostrar aquí su lugar. Hay una pequeña cosa mal con la visualización de la velocidad de cuadro que utiliza la variable `time_fac` para. Esta variable devuelve el valor 1, si la velocidad de cuadros es exactamente a 16 imágenes por segundo, y cambios proporcionalmente. Esto significa que la pantalla cambia con bastante rapidez y es difícil de leer. Un truco hará ralentizar un poco:

```
var fps;
function show_panel()
{
while (1) { // repeat forever
fps = 0.8*fps + 0.2*time_fac;
wait(1);
}
}
```

Nuestro eternamente ejecutando `show_panel` función es responsable de "procesamiento" de la pantalla de variables. Ello debe iniciarse en el juego por iniciar un `show_panel` instrucción dentro de la función principal y, a continuación, se correr para siempre. El `debug_panel` debe ahora mostrar los fps en lugar de la `time_fac` variable. El expresión aritmética se asegura de que un espontáneo cambio de `time_fac` sólo tiene un impacto de 20% en la pantalla, lo que hace que el valor para cambiar cinco veces más lento. Otro ejemplo de muestra es un mapa de bits que puede

ser cambiado en virtud de una ventana de acuerdo con la los valores de las variables dado. Nos gustaría hacer uso de una ventana de visualización para representar a una brújula en la pantalla. Empezamos por una pintura de mapa de bits de la brújula-bar (sustituir la 'O' por una 'E' para adaptarla a la el idioma Inglés):

bmap compass_map = <compass.pcx>; // 160x20 pixels	
var compass_pos[2] = 0,0;	
panel compass_pan {	
pos_x = 220;	
pos_y = 2;	
	flags = visible,refresh;
	window 0,0, 40,20,compass_map,compass_pos.x,compass_pos.y;
}	

La brújula siempre nos dan la cámara de línea de vista, es igual a 0 ° este. La ventana de línea de estructurado de manera similar a los dígitos de nuevo aquí en primer lugar, la posición se da, entonces la anchura y la altura de la ventana en píxeles y, a continuación, el mapa de bits a ser desplazado y, por último, los dos parámetros que dan la horizontal y vertical cambio en píxeles. Estamos utilizando sólo un cambio horizontal aquí, así compass_pos.y es siempre 0. Estamos compass_pos.x de calcular el ángulo de la cámara:

function show_panel()
{
while (1) { // repeat forever
fps = 0.8*fps + 0.2*time_fac;
compass_pos.x = 120 - (camera.pan % 360)/3;
wait(1);
}
}

Para el desplazamiento que hicimos alrededor de una brújula de mapa de bits 160 píxeles de ancho que se repite después de 120 píxeles. Por lo tanto, nuestra ventana de 40 píxeles de tamaño siempre es llenado con la brújula de mapa de bits (120 +40 = 160).

Para una rotación completa de 360 grados al mapa de bits debe recaer bei 120 píxeles, que la tercera - de modo que dividir el ángulo por tres. Tenemos que limitar la gama ángulo a 0 .. 360 por el módulo (%) función, porque camera.pan puede ser fuera de este rango. Y, finalmente, hemos invertido el cambio de dirección restando el importe de cambio de su valor máximo de 120 píxeles.