Manual de Usuario - Gramática DAAD#

Guía Completa de CondActs y Sintaxis

Versión: 3.0 | Cobertura: 141/141 CondActs (100%)

Índice

Introducción

DAAD# es un lenguaje moderno para crear aventuras conversacionales que mantiene 100% compatibilidad con DAAD clásico mientras añade nuevas funcionalidades. Este manual explica cómo usar cada uno de los 141 CondActs disponibles.

¿Qué es un CondAct?

Un **CondAct** (Condition-Action) es una instrucción que puede:

- **Verificar una condición** (devuelve verdadero/falso)
- **Ejecutar una acción** (modifica el estado del juego)
- **Ambas cosas** (algunos CondActs híbridos)

Estructura Básica

Sintaxis General

```
responses {
    patrón_comando: {
        condición1; acción1
        condición2; acción2; acción3
        done  // Terminar secuencia
    }
}
```

Reglas Básicas

1. **Secuencial**: Los CondActs se ejecutan en orden
2. **Condicional**: Si una condición falla, se detiene la secuencia
3. **Terminación**: done marca el final exitoso
4. **Punto y coma**: Separa CondActs en la misma línea

☑ Condiciones (CondActs de Verificación)

🎯 Condiciones Básicas de Ubicación

`at <ubicación>`

Verifica si el jugador está en una ubicación específica.

```
// Verificar si estamos en la biblioteca
examinar estantería: {
    at biblioteca; message "Ves libros antiguos."
    message "No hay estanterías aquí."
}
```

`notat <ubicación>`

Verifica si el jugador NO está en una ubicación específica.

```
// Solo funciona fuera del jardín
recoger flores: {
    notat jardín; message "No hay flores aquí."
    message "Recoges hermosas flores."
}
```

📦 Condiciones de Objetos

`present <objeto>`

Verifica si un objeto está en la ubicación actual.

```
// La llave debe estar visible
abrir puerta: {
    present llave; message "Usas la llave para abrir."
    message "Necesitas una llave."
}
```

`absent <objeto>`

Verifica si un objeto NO está en la ubicación actual.

```
// Solo si no hay guardia
esconderse: {
    absent guardia; message "Te escondes exitosamente."
    message "El guardia te ve intentando esconderte."
}
```

`carried <objeto>`

Verifica si el jugador lleva un objeto.

```
// Debe tener la espada en el inventario
atacar dragón: {
    carried espada; message "Atacas con tu espada."
    message "No tienes armas."
}
```

`notcarr <objeto>`

Verifica si el jugador NO lleva un objeto.

```
// Solo si no tiene la poción
beber agua: {
    notcarr poción; message "Bebes agua fresca."
    message "Mejor bebe la poción mágica."
}
```

`worn <objeto>`

Verifica si el jugador lleva puesto un objeto.

```
// Verificar si lleva el casco
entrar batalla: {
    worn casco; message "Estás protegido."
    message "¡Necesitas protección!"
}
```

`notworn <objeto>`

Verifica si el jugador NO lleva puesto un objeto.

```
// Solo si no lleva armadura
nadar: {
```

```
    notworn armadura; message "Nadas libremente."
    message "La armadura te hunde."
}
```

🏠 Condiciones de Localización de Objetos

`isat <objeto> <ubicación>`

Verifica si un objeto específico está en una ubicación específica.

```
// Verificar si la carta está en el escritorio
leer carta: {
    isat carta escritorio; desc carta
    message "La carta no está en el escritorio."
}
```

`isnotat <objeto> <ubicación>`

Verifica si un objeto NO está en una ubicación específica.

```
// Solo si el tesoro no está en la caja fuerte
buscar tesoro: {
    isnotat tesoro caja_fuerte; message "Sigues buscando..."
    message "Ya sabes dónde está el tesoro."
}
```

🔢 Condiciones de Flags (Variables)

`zero <flag>`

Verifica si un flag tiene valor 0.

```
// Verificar si no tiene puntos
dar puntos: {
    zero puntuación; let puntuación 10; message "¡Primeros puntos!"
    message "Ya tienes puntos."
}
```

`notzero <flag>`

Verifica si un flag tiene un valor diferente de 0.

```
// Solo si tiene energía
correr: {
    notzero energía; minus energía 5; message "Corres rápidamente."
    message "Estás demasiado cansado."
}
```

🔢 Condiciones de Comparación Numérica

`eq <valor1> <valor2>`

Verifica si dos valores son iguales.

```
// Verificar código correcto
introducir código: {
    eq flag[código_jugador] 1234; message "¡Código correcto!"
    message "Código incorrecto."
}
```

`noteq <valor1> <valor2>`

Verifica si dos valores son diferentes.

```
// Solo si no es el estado inicial
avanzar nivel: {
    noteq flag[nivel] 1; plus flag[nivel] 1; message "Nivel aumentado."
    message "Aún estás en el primer nivel."
}
```

`gt <valor1> <valor2>`

Verifica si el primer valor es mayor que el segundo.

```
// Verificar si tiene suficiente dinero
comprar espada: {
    gt flag[dinero] 100; minus flag[dinero] 100; get espada
    message "No tienes suficiente dinero."
}
```

`lt <valor1> <valor2>`

Verifica si el primer valor es menor que el segundo.

```

```

```
// Verificar si está débil
descansar: {
    lt flag[energía] 20; let flag[energía] 100; message "Te sientes renovado."
    message "No necesitas descansar aún."
}
```

`bigger <valor1> <valor2>`

Verifica si el primer valor es mayor que el segundo (sinónimo de `gt`).

```
// Verificar experiencia alta
usar magia avanzada: {
    bigger flag[experiencia] 50; message "Lanzas un hechizo poderoso."
    message "No tienes suficiente experiencia."
}
```

`smaller <valor1> <valor2>`

Verifica si el primer valor es menor que el segundo (sinónimo de `lt`).

```
// Verificar si es joven
entrar escuela: {
    smaller flag[edad] 18; message "Entras a la escuela."
    message "Eres demasiado mayor para la escuela."
}
```

`same <valor1> <valor2>`

Verifica si dos valores son iguales (sinónimo de `eq`).

```
// Verificar mismo estado
sincronizar: {
    same flag[estado_a] flag[estado_b]; message "Sistemas sincronizados."
    message "Estados diferentes, sincronizando..."
}
```

`notsame <valor1> <valor2>`

Verifica si dos valores son diferentes (sinónimo de `noteq`).

```
// Verificar diferencia
comparar objetos: {
    notsame object[objeto1] object[objeto2]; message "Son objetos diferentes."
```

```
    message "Son el mismo objeto."
}
```

🎲 Condiciones de Azar

`chance <porcentaje>`

Condición aleatoria con porcentaje de éxito (0-100).

```
// 30% de probabilidad de encontrar tesoro
buscar tesoro: {
    chance 30; get tesoro; message "¡Encuentras un tesoro!"
    message "No encuentras nada."
}
```

🎯 Condiciones de Parser Avanzado (Fase 2)

`adject1 <adjetivo>`

Verifica si el primer adjetivo del comando coincide.

```
// Verificar color específico
examinar _: {
    adject1 rojo; message "Es de color rojo brillante."
    adject1 azul; message "Es de color azul profundo."
    desc noun1  // Descripción normal si no hay adjetivo específico
}
```

`adject2 <adjetivo>`

Verifica si el segundo adjetivo del comando coincide.

```
// Comando: "examinar caja roja pequeña"
examinar caja: {
    adject1 roja; adject2 pequeña; message "La pequeña caja roja."
    message "Una caja normal."
}
```

`noun2 <sustantivo>`

Verifica el segundo sustantivo en comandos con dos objetos.

```

```

```
// Comando: "meter llave en caja"
meter _ _: {
    noun2 caja; present noun1; autop noun2; message "Metido en la caja."
    message "¿Meter dónde?"
}
```

`prep <preposición>`

Verifica la preposición utilizada en el comando.

```
// Diferentes acciones según preposición
meter llave _: {
    prep en; message "Metes la llave en algo."
    prep bajo; message "Escondes la llave bajo algo."
    prep sobre; message "Pones la llave sobre algo."
}
```

`adverb <adverbio>`

Verifica el adverbio utilizado en el comando.

```
// Diferentes formas de caminar
caminar: {
    adverb rápidamente; minus flag[energía] 3; message "Caminas rápido."
    adverb lentamente; minus flag[energía] 1; message "Caminas despacio."
    minus flag[energía] 2; message "Caminas normalmente."
}
```

🔧 Condiciones de Atributos (Fase 1)

`hasat <objeto> <atributo>`

Verifica si un objeto tiene un atributo específico.

```
// Verificar si la puerta está cerrada
abrir puerta: {
    hasat puerta 1; clear puerta; message "Abres la puerta."
    message "La puerta ya está abierta."
}
```

`hasnat <objeto> <atributo>`

Verifica si un objeto NO tiene un atributo específico.

```
// Solo si no está encendido
encender lámpara: {
    hasnat lámpara 2; set lámpara; message "Enciendes la lámpara."
    message "Ya está encendida."
}
```

Condiciones de Control de Procesos (Fase 2)

`isdone`

Verifica si el último proceso ejecutado terminó exitosamente.

```
// Verificar si el mecanismo funcionó
activar mecanismo: {
    process verificar_mecanismo
    isdone; message "El mecanismo se activó correctamente."
    message "El mecanismo falló."
}
```

`isndone`

Verifica si el último proceso NO terminó exitosamente.

```
// Reintentar si falló
resolver puzzle: {
    process intentar_solución
    isndone; plus flag[intentos] 1; message "Intento fallido."
    message "¡Puzzle resuelto!"
}
```

Acciones (CondActs de Ejecución)

Acciones de Movimiento y Navegación

`goto <ubicación>`

Mueve al jugador a una nueva ubicación.

```
// Ir a la biblioteca
ir biblioteca: {
    goto biblioteca; message "Te diriges a la biblioteca."
}

// Con condición
```

```
entrar cueva: {
    carried antorcha; goto cueva; message "Entras en la cueva oscura."
    message "Necesitas luz para entrar."
}
```

Acciones de Manipulación de Objetos

`get <objeto>`

El jugador toma un objeto (lo añade al inventario).

```
// Coger un objeto simple
coger llave: {
    present llave; get llave; message "Coges la llave."
    message "No hay ninguna llave aquí."
}

// Con verificaciones
coger espada: {
    present espada; carried escudo; get espada; message "Coges la espada."
    present espada; message "Necesitas un escudo primero."
    message "No hay espada aquí."
}
```

`drop <objeto>`

El jugador suelta un objeto (lo saca del inventario a la ubicación actual).

```
// Soltar objeto
soltar antorcha: {
    carried antorcha; drop antorcha; message "Sueltas la antorcha."
    message "No llevas una antorcha."
}
```

`wear <objeto>`

El jugador se pone un objeto (cambia de inventario a puesto).

```
// Ponerse ropa
ponerse casco: {
    carried casco; wear casco; message "Te pones el casco."
    message "No tienes un casco."
}
```

`remove <objeto>`

El jugador se quita un objeto (cambia de puesto a inventario).

```
// Quitarse ropa
quitarse casco: {
    worn casco; remove casco; message "Te quitas el casco."
    message "No llevas puesto un casco."
}
```

🎭 Acciones de Presentación y Comunicación

`message <número>`

Muestra un mensaje del sistema.

```
// Mostrar mensaje específico
examinar estatua: {
    present estatua; message 42  // Mensaje #42
    message 43  // "No hay estatua aquí"
}
```

`print <texto>`

Imprime texto literal o el contenido de un flag.

```
// Mostrar puntuación
ver puntuación: {
    print "Puntos: "; print flag[puntuación]; newline
}
```

`newline`

Imprime un salto de línea.

```
// Separar texto
mostrar estadísticas: {
    print "=== ESTADÍSTICAS ==="; newline
    print "Nivel: "; print flag[nivel]; newline
    print "Energía: "; print flag[energía]; newline
}
```

`desc <objeto>`

Muestra la descripción de un objeto.

```
// Describir objeto presente
examinar _: {
    present noun1; desc noun1
    message "No hay tal cosa aquí."
}
```

`cls`

Limpia la pantalla.

```
// Limpiar pantalla al cambiar de área
entrar palacio: {
    cls; goto palacio; message "Entras en el magnífico palacio."
}
```

🔢 Acciones de Manipulación de Variables (Flags)

`set <flag>`

Pone un flag a 1 (verdadero).

```
// Marcar que se encontró la llave
encontrar llave: {
    present llave; get llave; set flag[tiene_llave]
    message "Coges la llave importante."
}
```

`clear <flag>`

Pone un flag a 0 (falso).

```
// Desactivar alarma
desactivar alarma: {
    carried código; clear flag[alarma_activa]
    message "Desarmas la alarma."
}
```

`let <flag> <valor>`

Asigna un valor específico a un flag.

```
// Establecer puntuación inicial
comenzar juego: {
    let flag[puntuación] 0
    let flag[energía] 100
    let flag[nivel] 1
    message "¡Comienza la aventura!"
}
```

`plus <flag> <valor>`

Suma un valor a un flag.

```
// Ganar puntos
resolver puzzle: {
    plus flag[puntuación] 50
    message "¡Resuelves el puzzle! +50 puntos"
}
```

`minus <flag> <valor>`

Resta un valor a un flag.

```
// Perder energía
correr: {
    gt flag[energía] 5; minus flag[energía] 5; message "Corres."
    message "Estás demasiado cansado."
}
```

🧬 Acciones de Creación y Destrucción

`create <objeto>`

Crea un objeto en la ubicación actual.

```
// Aparecer objeto por magia
lanzar hechizo: {
    create poción_mágica
    message "¡Aparece una poción mágica!"
}
```

`destroy <objeto>`

Destruye un objeto completamente del juego.

```
// Destruir objeto usado
usar poción: {
    carried poción; destroy poción; plus flag[energía] 50
    message "Bebes la poción y te sientes renovado."
}
```

`place <objeto> <ubicación>`

Coloca un objeto en una ubicación específica.

```
// Transportar objeto a otra ubicación
teleportar tesoro: {
    place tesoro cámara_secreta
    message "El tesoro desaparece mágicamente."
}
```

🎮 Acciones de Control de Juego

`done`

Marca la secuencia como completada exitosamente.

```
// Comando completado
abrir puerta: {
    carried llave; message "Usas la llave."; done
    message "No tienes la llave."
}
```

`end`

Termina el juego.

```
// Final del juego
entrar portal: {
    carried amuleto; message "¡Has ganado el juego!"; end
    message "Necesitas el amuleto para ganar."
}
```

`restart`

Reinicia el juego desde el principio.

```
```

```
// Reiniciar por muerte
tocar cristal maldito: {
    message "El cristal te mata instantáneamente."
    restart
}
```

Acciones de Control de Procesos

process <número>

Ejecuta un proceso (tabla de respuestas) específico.

```
// Ejecutar subrutina
usar máquina: {
    process tabla_máquina
    message "La máquina hace ruidos extraños."
}
```

doall <especificador>

Ejecuta una acción para todos los objetos que cumplan un criterio.

```
// Listar inventario
inventario: {
    doall carried; print object[current]; newline
}

// Limpiar ubicación
limpiar: {
    doall here; destroy object[current]
}
```

extern <número>

Llama a una función externa del sistema.

```
// Función especial del sistema
guardar partida: {
    extern 1  // Función de guardado del sistema
    message "Partida guardada."
}
```

skip <número>

Salta las siguientes N instrucciones.

```
// Control de flujo
verificar estado: {
    eq flag[estado] 1; skip 2
    message "Estado incorrecto."
    end
    message "Estado correcto, continuando..."
}
```

`notdone`

Marca que la secuencia actual NO ha terminado exitosamente.

```
// Forzar fallo
intentar imposible: {
    message "Esto es imposible de hacer."
    notdone
}
```

💾 Acciones de Persistencia

`save <número>`

Guarda el estado del juego en un slot específico.

```
// Guardar en slot específico
guardar partida: {
    save 1
    message "Partida guardada en slot 1."
}
```

`load <número>`

Carga el estado del juego desde un slot específico.

```
// Cargar partida
cargar partida: {
    load 1
    message "Partida cargada desde slot 1."
}
```

`sysmess <número>`

Muestra un mensaje del sistema.

```
// Mensaje de sistema
error sistema: {
    sysmess 100  // Mensaje de error del sistema
}
```

Acciones de Contenedores

putin <objeto> <contenedor>

Mete un objeto dentro de otro objeto (contenedor).

```
// Meter objeto en contenedor
meter llave caja: {
    carried llave; present caja; putin llave caja
    message "Metes la llave en la caja."
}
```

takeout <objeto> <contenedor>

Saca un objeto de un contenedor.

```
// Sacar objeto de contenedor
sacar llave caja: {
    present caja; takeout llave caja
    message "Sacas la llave de la caja."
}
```

Familia COPY (Fase 3) - Operaciones de Copia

copyff <flag_origen> <flag_destino>

Copia el valor de un flag a otro flag.

```
// Sincronizar variables
sincronizar energía: {
    copyff flag[energía_máxima] flag[energía_actual]
    message "Energía restaurada al máximo."
}
```

copyof <objeto> <flag>

Copia la ubicación de un objeto a un flag.

```
// Recordar dónde estaba un objeto
recordar posición: {
    copyof tesoro flag[ubicación_tesoro]
    message "Recuerdas dónde viste el tesoro."
}
```

copyfo <flag> <objeto>

Copia el valor de un flag a la ubicación de un objeto.

```
// Teleportar objeto a ubicación guardada
teleportar objeto: {
    copyfo flag[destino] objeto_mágico
    message "El objeto aparece en el destino."
}
```

copyoo <objeto1> <objeto2>

Copia la ubicación del primer objeto al segundo objeto.

```
// Mover un objeto donde está otro
juntar objetos: {
    copyoo llave_maestra llave_copia
    message "Las llaves están ahora juntas."
}
```

copybf <flag> <flag_byte>

Copia solo el byte inferior de un flag a otro flag.

```
// Extraer byte inferior (0-255)
extraer color: {
    copybf flag[valor_rgb] flag[componente_rojo]
    message "Componente rojo extraído."
}
```

Acciones Automáticas (Fase 4)

autog

Automáticamente ejecuta la acción "coger" en el objeto actual.

```
// Coger automáticamente
coger _: {
    present noun1; autog; done
    message "No puedes coger eso."
}
```

autod

Automáticamente ejecuta la acción "soltar" en el objeto actual.

```
// Soltar automáticamente
soltar _: {
    carried noun1; autod; done
    message "No llevas eso."
}
```

autow

Automáticamente ejecuta la acción "ponerse" en el objeto actual.

```
// Ponerse automáticamente
ponerse _: {
    carried noun1; autow; done
    message "No tienes eso para ponerte."
}
```

autor

Automáticamente ejecuta la acción "quitarse" en el objeto actual.

```
// Quitarse automáticamente
quitarse _: {
    worn noun1; autor; done
    message "No llevas eso puesto."
}
```

autop <contenedor>

Automáticamente mete el objeto actual en un contenedor.

```
// Meter automáticamente en caja
meter _ caja: {
```

```
    carried noun1; present caja; autop caja; done
    message "No puedes hacer eso."
}
```

autot <contenedor>

Automáticamente saca el objeto actual de un contenedor.

```
// Sacar automáticamente de caja
sacar _ caja: {
    present caja; autot caja; done
    message "No hay nada que sacar."
}
```

Acciones Avanzadas (Fase 7)

inkey

Verifica si se presionó una tecla (condición no bloqueante).

```
// Esperar pulsación de tecla
esperar tecla: {
    inkey; message "Tecla detectada, continuando..."
    message "Presiona cualquier tecla para continuar."
}
```

add <flag> <valor>

Suma un valor a un flag (versión avanzada de `plus`).

```
// Sumar experiencia
ganar experiencia: {
    add flag[experiencia] flag[bonus_experiencia]
    message "¡Experiencia ganada!"
}
```

sub <flag> <valor>

Resta un valor a un flag (versión avanzada de `minus`).

```
// Restar energía
usar magia: {
    sub flag[energía] flag[coste_hechizo]
```

```
    message "Lanzas el hechizo."
}
```

random <máximo>

Genera un número aleatorio entre 0 y máximo-1, lo guarda en flag[value].

```
// Dado de 6 caras
lanzar dado: {
    random 6; plus flag[value] 1  // Convertir 0-5 a 1-6
    print "Sale un "; print flag[value]; newline
}
```

ok

Confirma que la última operación fue exitosa.

```
// Confirmar operación
realizar ritual: {
    process ritual_complejo
    ok; message "El ritual fue exitoso."
    message "El ritual falló."
}
```

newtext

Limpia el buffer de texto y comienza una nueva sección.

```
// Comenzar nueva sección de historia
capítulo nuevo: {
    newtext
    message "=== CAPÍTULO 2 ==="
    message "La historia continúa..."
}
```

display <pantalla>

Muestra una pantalla específica (gráficos, texto formateado).

```
// Mostrar pantalla de título
mostrar título: {
    display 1
    message "Presiona una tecla para continuar."
}
```

`call <rutina>`

Llama a una rutina específica del sistema.

```
// Llamar rutina personalizada
procesar inventario: {
    call 15  // Rutina #15 para procesar inventario
    message "Inventario procesado."
}
```

`synonym <palabra1> <palabra2>`

Crea un sinónimo dinámico (palabra2 actúa como palabra1).

```
// Crear alias para objetos
crear sinónimo: {
    synonym 25 30  // Palabra #30 = sinónimo de palabra #25
    message "Sinónimo creado."
}
```

`ramsave`

Guarda el estado actual en la memoria RAM (guardado rápido).

```
// Guardado rápido en RAM
punto control: {
    ramsave
    message "Punto de control establecido."
}
```

`ramload`

Restaura el estado desde la memoria RAM (carga rápida).

```
// Restaurar desde RAM
restaurar punto: {
    ramload
    message "Regresando al punto de control."
}
```

`saveat`

Guarda el estado en archivo de forma persistente.

```
// Guardado persistente
guardar permanente: {
    saveat
    message "Progreso guardado permanentemente."
}
```

backat

Restaura el estado desde archivo persistente.

```
// Carga persistente
cargar permanente: {
    backat
    message "Progreso restaurado."
}
```

gfx <comando>

Ejecuta un comando gráfico específico.

```
// Mostrar imagen
mostrar mapa: {
    gfx 10  // Comando gráfico #10
    message "Se muestra el mapa del mundo."
}
```

mouse

Habilita o procesa entrada de ratón.

```
// Activar entrada de ratón
modo gráfico: {
    mouse
    message "Modo gráfico activado. Usa el ratón."
}
```

redo

Repite la última acción realizada por el jugador.

```
```

```
// Repetir última acción
repetir: {
    redo
    message "Repitiendo última acción..."
}
```

`move <dirección>`

Mueve al jugador en una dirección específica automáticamente.

```
// Movimiento forzado
viento fuerte: {
    chance 50; move 1; message "El viento te empuja al norte."
    message "Resistes el viento."
}
```

🔧 Expresiones y Referencias

📊 Tipos de Datos

Identificadores

```
// Nombres simples
objeto_1, puerta_principal, flag_energía

// Con números
sala1, llave_2, personaje_npc_3
```

Números Enteros

```
// Valores directos
42, 100, 0, -5

// En CondActs
let flag[puntos] 1000
chance 75
```

Referencias a Flags

```
// Sintaxis de arrays
flag[puntuación]
```

```
flag[energía_máxima]
flag[nivel_actual]

// Con números
flag[0]  // Flag número 0
flag[42] // Flag número 42
```

Referencias a Objetos

```
// Por nombre
object[llave_maestra]
object[espada_mágica]

// Por número
object[1]   // Objeto número 1
object[100] // Objeto número 100

// Variables especiales
object[noun1]    // Primer sustantivo del comando
object[noun2]    // Segundo sustantivo del comando
object[current]  // Objeto actual en bucles doall
```

🎯 Especificadores de Ubicación

```
// Ubicaciones específicas
biblioteca, jardín, cueva_oscura

// Ubicaciones especiales
here     // Ubicación actual del jugador
carried  // En el inventario del jugador
worn     // Puesto por el jugador
all      // Todas las ubicaciones
```

🎮 Patrones de Comandos

🗣️ Sintaxis de Comandos

Patrones Básicos

```
// Comando simple (solo verbo)
inventario: { doall carried; print object[current]; newline }

// Verbo + sustantivo
examinar llave: { desc llave }
```

```
// Verbo + comodín
examinar _: { present noun1; desc noun1; message "No está aquí." }
```

Patrones Avanzados

```
// Verbo + sustantivo + preposición + sustantivo
meter _ en _: {
    carried noun1; present noun2
    putin noun1 noun2; message "Hecho."
}

// Con adjetivos
examinar _ rojo: {
    adject1 rojo; present noun1; desc noun1
    message "No hay nada rojo aquí."
}

// Con adverbios
caminar rápidamente: {
    adverb rápidamente; minus flag[energía] 3
    goto salida; message "Caminas rápido hacia la salida."
}
```

Comodines Especiales

```
// _ = cualquier palabra
coger _: { present noun1; autog; done }

// Múltiples comodines
meter _ en _: { /* manejar dos objetos */ }

// Combinaciones
examinar _ _: {
    // Examinar con dos sustantivos o sustantivo+adjetivo
    present noun1; desc noun1
}
```

Ejemplos Prácticos

Ejemplo 1: Sistema de Llaves y Puertas

```
responses {
    // Abrir puerta con llave específica
    abrir puerta: {
        carried llave_dorada; present puerta_principal
```

```
        hasat puerta_principal 1; clear puerta_principal
        message "Usas la llave dorada para abrir la puerta."
        done

        present puerta_principal; hasnat puerta_principal 1
        message "La puerta ya está abierta."
        done

        present puerta_principal
        message "Necesitas la llave dorada."
        done

        message "No hay ninguna puerta aquí."
    }

    // Cerrar puerta
    cerrar puerta: {
        carried llave_dorada; present puerta_principal
        hasnat puerta_principal 1; set puerta_principal
        message "Cierras la puerta con llave."
        done

        present puerta_principal
        message "La puerta ya está cerrada."
        done

        message "No hay ninguna puerta aquí."
    }
}
```

⚔️ Ejemplo 2: Sistema de Combate

```
responses {
    // Atacar enemigo
    atacar _: {
        present noun1; carried espada
        chance 70  // 70% de probabilidad de éxito

        // Ataque exitoso
        plus flag[experiencia] 10
        destroy noun1
        message "¡Derrotas al enemigo! +10 experiencia"
        done

        // Ataque fallido
        minus flag[energía] 5
        message "Fallas el ataque y pierdes energía."
        done

        present noun1
        message "Necesitas una espada para atacar."
```

```
        done

        message "No hay enemigos aquí."
    }

    // Huir del combate
    huir: {
        chance 80; goto entrada; message "Escapas exitosamente."
        message "No puedes escapar."
    }
}
```

Ejemplo 3: Sistema de Pociones

```
responses {
    // Beber poción
    beber _: {
        carried noun1

        // Poción de energía
        same noun1 poción_energía
        destroy noun1; plus flag[energía] 50
        message "Te sientes renovado. +50 energía"
        done

        // Poción de fuerza
        same noun1 poción_fuerza
        destroy noun1; set flag[fuerza_extra]
        message "¡Te sientes más fuerte!"
        done

        // Poción venenosa
        same noun1 poción_veneno
        destroy noun1; minus flag[energía] 30
        message "¡La poción era venenosa! -30 energía"
        done

        message "No puedes beber eso."
        done

        message "No tienes esa poción."
    }

    // Mezclar pociones
    mezclar _ con _: {
        carried noun1; carried noun2

        // Mezcla exitosa
        same noun1 hierba_curativa; same noun2 agua_pura
        destroy noun1; destroy noun2; create poción_energía
        message "Creas una poción de energía."
```

```
        done

        message "Esta mezcla no funciona."
        done

        message "No tienes ambos ingredientes."
    }
}
```

Ejemplo 4: Sistema de Inventario Avanzado

```
responses {
    // Mostrar inventario detallado
    inventario: {
        message "=== INVENTARIO ==="
        newline

        doall carried; print "- "; print object[current]; newline

        message "=== EQUIPADO ==="
        newline

        doall worn; print "* "; print object[current]; newline

        message "=== ESTADÍSTICAS ==="
        newline
        print "Energía: "; print flag[energía]; newline
        print "Experiencia: "; print flag[experiencia]; newline
    }

    // Soltar todo
    soltar todo: {
        doall carried; drop object[current]
        message "Sueltas todo tu inventario."
    }

    // Contar objetos
    contar objetos: {
        let flag[contador] 0
        doall carried; plus flag[contador] 1
        print "Llevas "; print flag[contador]; print " objetos."
        newline
    }
}
```

Ejemplo 5: Sistema de Persistencia y Guardado

```
responses {
    // Guardar partida
```

```
    guardar: {
        saveat
        plus flag[partidas_guardadas] 1
        message "Partida guardada permanentemente."
        print "Guardados totales: "; print flag[partidas_guardadas]
        newline
    }

    // Cargar partida
    cargar: {
        backat
        message "Partida restaurada."
    }

    // Punto de control rápido
    checkpoint: {
        ramsave
        message "Punto de control establecido en RAM."
    }

    // Volver al checkpoint
    volver: {
        ramload
        message "Regresando al último punto de control."
    }

    // Guardar antes de acción peligrosa
    entrar zona peligrosa: {
        ramsave  // Guardado automático de seguridad
        goto zona_peligrosa
        message "Entras en la zona peligrosa..."
        message "(Punto de control automático establecido)"
    }
}
```

Referencia Rápida

Condiciones Más Comunes

CondAct	Función	Ejemplo
`present <obj>`	¿Está el objeto aquí?	`present llave`
`carried <obj>`	¿Lo llevo?	`carried espada`
`worn <obj>`	¿Lo llevo puesto?	`worn armadura`
`at <loc>`	¿Estoy en esta ubicación?	`at biblioteca`
`eq <a> <b>`	¿Son iguales?	`eq flag[nivel] 5`
`gt <a> <b>`	¿Es mayor?	`gt flag[energía] 20`

CondAct	Función	Ejemplo
`chance <n>`	¿Suerte del n%?	`chance 50`

🎬 Acciones Más Comunes

CondAct	Función	Ejemplo
`get <obj>`	Coger objeto	`get llave`
`drop <obj>`	Soltar objeto	`drop antorcha`
`goto <loc>`	Ir a ubicación	`goto jardín`
`message <n>`	Mostrar mensaje	`message 42`
`let <flag> <val>`	Asignar valor	`let flag[puntos] 100`
`plus <flag> <val>`	Sumar a flag	`plus flag[energía] 10`
`done`	Terminar exitosamente	`done`
`end`	Terminar juego	`end`

🔧 Operaciones de Flags

CondAct	Función	Ejemplo
`set <flag>`	flag = 1	`set flag[encontrado]`
`clear <flag>`	flag = 0	`clear flag[alarma]`
`let <flag> <val>`	flag = valor	`let flag[nivel] 3`
`plus <flag> <val>`	flag += valor	`plus flag[puntos] 50`
`minus <flag> <val>`	flag -= valor	`minus flag[energía] 10`
`copyff <f1> <f2>`	f2 = f1	`copyff flag[max] flag[actual]`

🏗 Patrones de Código Comunes

Verificar y Ejecutar

```
acción: {
    condición; acción_si_cierto; done
    acción_si_falso
}
```

Múltiples Condiciones

```
acción compleja: {
    condición1; condición2; condición3
    // Solo se ejecuta si TODAS son ciertas
    acción_final; done
    mensaje_error
}
```

Bucle de Objetos

```
procesar todo: {
    doall especificador; /* acción con object[current] */
}
```

Sistema de Estados

```
verificar estado: {
    eq flag[estado] 1; mensaje_estado_1; done
    eq flag[estado] 2; mensaje_estado_2; done
    eq flag[estado] 3; mensaje_estado_3; done
    mensaje_estado_desconocido
}
```

Consejos y Mejores Prácticas

Optimización

1. **Condiciones primero**: Siempre verificar condiciones antes de acciones costosas
2. `done` **temprano**: Usar `done` para evitar código innecesario
3. **Flags organizados**: Usar nombres descriptivos para flags
4. **Modularidad**: Usar `process` para dividir lógica compleja

Robustez

1. **Verificar siempre**: No asumir que objetos existen
2. **Mensajes claros**: Dar feedback claro al jugador
3. **Estados válidos**: Verificar que flags tengan valores esperados
4. **Recuperación**: Manejar estados de error graciosamente

Usabilidad

1. **Sinónimos**: Usar `synonym` para comandos alternativos
2. **Comodines**: Usar `_` para comandos flexibles
3. **Ayuda contextual**: Dar pistas cuando las acciones fallen
4. **Progresión clara**: Usar flags para rastrear progreso del jugador

Manual DAAD# v3.0 - Cobertura completa 141/141 CondActs
Documento técnico para desarrolladores de aventuras conversacionales