team banzai

DOCUMENTACIÓN TÉCNICA DE ARQUITECTURA (ARC42)

El ciclo de una puerta

Subsistema de puertas de DOOM (linuxdoom-1.10), para quien mantiene el código

Generado desde el código fuente, commit a77dfb9

16 de julio de 2026

Team Banzai S.L.U. · Demo de documentación viva sobre DOOM

> **Sobre este documento.** Esta documentación sigue **arc42**, el estándar de facto para documentar arquitectura de software: incorpora vistas al estilo **C4** (mapa de módulos, vista de bloques y vista en ejecución) y una decisión en formato **ADR**. Por ser un subsistema, se usan los apartados relevantes de arc42; los que no aplican a esta escala (estrategia de solución, despliegue, conceptos transversales) se omiten de forma deliberada. Cada afirmación cuantitativa lleva su referencia `archivo:línea`, su confianza y el método de comprobación.

Índice

1. Introducción y objetivos

Este documento describe el recorrido completo de una puerta de DOOM, de principio a fin: cómo se dispara (por usarla, por pisar una línea o por accionar un interruptor), la estructura de estado que la representa, la máquina de estados que la anima tic a tic, el motor de movimiento que reutiliza, el sonido, el rebote cuando atrapa a alguien debajo, las constantes con su unidad y su conversión a segundos, las funciones implicadas, los puntos por donde tocarla sin romper nada y los casos límite que el código ya contempla. Al final, tres constantes de daño y munición que no son de la puerta pero forman parte del examen.

> **Disciplina de esta documentación.** Se generó leyendo **exclusivamente** `codigo-mutado/linuxdoom-1.10/`, commit de corte `a77dfb96cb91780ca334d0d4cfd86957558007e0`. Los valores reportados son los que hay en el código a la vista; no se afirma ningún número que no esté en una línea leída. Cuando se cita un valor de la build pública (la que documenta la Doom Wiki), se marca como contraste, nunca como la verdad de este árbol.

3. Contexto y alcance

3.1 Las tres vías de activación

Una puerta puede nacer de tres formas distintas. Las tres acaban creando el mismo `vldoor_t` y poniéndolo bajo el control de `T_VerticalDoor`, pero por rutas separadas:

1. **Por usarla (puerta manual, sin etiqueta).** El jugador pulsa «usar». `P_PlayerThink` comprueba el botón `BT_USE` `p_user.c:322` y llama a `P_UseLines` `p_user.c:326`. Esta traza una línea corta hacia delante `p_map.c:1130` y, si encuentra una línea especial, la despacha con `P_UseSpecialLine` `p_map.c:1119`. Ese despachador, para los tipos de puerta manual (specials 1, 26, 27, 28, 31, 32, 33, 34, 117, 118), llama a `EV_VerticalDoor(line, thing)` `p_switch.c:323-338`. confianza: alta
2. **Por pisar una línea (walkover, puerta remota).** `P_CrossSpecialLine` `p_spec.c:492` se dispara al cruzar ciertas líneas. Los tipos 2 (abrir), 3 (cerrar) y 4 (subir) llaman a `EV_DoDoor` con el `vldoor_e` correspondiente `p_spec.c:545,551,557` y anulan el special de la línea para que no se repita. confianza: alta
3. **Por interruptor.** `P_UseSpecialLine` `p_switch.c:276`, más abajo del bloque de puertas manuales, cablea numerosos interruptores a `EV_DoDoor` `p_switch.c:403,457,463` y a `EV_DoLockedDoor` para los interruptores con llave `p_switch.c:503,634`. confianza: alta

La diferencia práctica: `EV_VerticalDoor` opera sobre el sector al otro lado de **esa misma** línea y admite reabrir o invertir una puerta ya activa; `EV_DoDoor` opera sobre **todos** los sectores cuya etiqueta coincide con la de la línea `p_doors.c:275` y omite los que ya tienen una puerta en marcha `p_doors.c:278-279`.

3.2 Mapa de módulos

Fichero	Papel en el recorrido de la puerta	Evidencia
`p_user.c`	Cada tic, si el jugador pulsa «usar», lanza el trazado: `P_UseLines`.	`p_user.c:322,326`

Fichero	Papel en el recorrido de la puerta	Evidencia
`p_map.c`	Traza la línea frente al jugador y despacha la línea especial encontrada.	`p_map.c:1119,1130`
`p_switch.c`	Despacha por `line->special`; para puertas manuales invoca `EV_VerticalDoor`.	`p_switch.c:276,338`
`p_spec.c`	Puertas remotas al pisar o disparar una línea; llaman a `EV_DoDoor`.	`p_spec.c:492,545`
`p_doors.c`	Toda la lógica de puertas: creación y el «thinker» (rutina por tic).	`p_doors.c:63,353,263`
`p_spec.h`	La estructura `vldoor_t`, el enum `vldoor_e` y las constantes de puerta.	`p_spec.h:328-365`
`p_floor.c`	`T_MovePlane`, el motor de movimiento vertical que la puerta reutiliza.	`p_floor.c:49`
`doomdef.h`	`TICRATE = 35` tics por segundo, la base de toda conversión a tiempo.	`doomdef.h:122`
`m_fixed.h`	`FRACUNIT = 65536`, la unidad de coma fija (una unidad de mapa).	`m_fixed.h:35-36`
`sounds.h`	Los identificadores de sonido que la puerta dispara.	`sounds.h:199-268`

Confianza global del mapa: alta; método: seguir cada llamada citada de fichero en fichero.

5. Vista de bloques de construcción

5.1 Estructura de estado: `vldoor_t`

Definida en `p_spec.h:343-360`. Es lo único que persiste entre tics; el thinker no guarda estado local. confianza: alta

Campo	Tipo	Significado	Evidencia
`thinker`	`thinker_t`	nodo en la lista de thinkers; lo enlaza `P_AddThinker`	`p_spec.h:345`
`type`	`vldoor_e`	variante de puerta (ver tabla siguiente)	`p_spec.h:346`
`sector`	`sector_t*`	el sector cuyo techo mueve la puerta	`p_spec.h:347`
`topheight`	`fixed_t`	altura de techo a la que se detiene arriba	`p_spec.h:348`
`speed`	`fixed_t`	unidades de mapa por tic que recorre	`p_spec.h:349`
`direction`	`int`	1 sube, 0 espera arriba, −1 baja, 2 espera inicial	`p_spec.h:351-352`
`topwait`	`int`	tics a esperar arriba antes de cerrar	`p_spec.h:354-355`
`topcountdown`	`int`	contador vivo; al llegar a 0 arranca el cierre	`p_spec.h:356-358`

Nota sobre `direction`: el comentario del código solo enumera 1/0/-1 `p_spec.h:351`, pero el thinker usa además el valor 2 como «espera inicial» `p_doors.c:99`, fijado únicamente por `P_SpawnDoorRaiseIn5Mins` `p_doors.c:543`. Es un estado real no documentado en el comentario. confianza: alta

5.2 Los ocho tipos de puerta: `vldoor_e`

El enum `p_spec.h:328-339` tiene ocho variantes. El `type` decide qué hace la puerta al llegar arriba y al llegar abajo:

type	Al llegar arriba	Al llegar abajo	Velocidad
`normal`	espera `topwait` y baja `p_doors.c:181-183`	se retira, libera el sector `p_doors.c:136-140`	`VDOORSPEED`
`open`	se retira, queda abierta `p_doors.c:185-190`	(no baja sola)	`VDOORSPEED`
`close`	(no sube)	se retira, y no rebota si aplasta `p_doors.c:155-157`	`VDOORSPEED`
`close30ThenOpen`	(sube tras 30 s abajo)	espera `35*30` tics y reabre `p_doors.c:142-145`	`VDOORSPEED`
`raiseIn5Mins`	como `normal` al arrancar	como `normal`	`VDOORSPEED`
`blazeRaise`	espera y baja, sonido propio `p_doors.c:75-79`	se retira, `sfx_bdcls` `p_doors.c:128-134`	`VDOORSPEED*4`
`blazeOpen`	se retira, queda abierta `p_doors.c:186-190`	(no baja sola)	`VDOORSPEED*4`
`blazeClose`	(no sube)	se retira, `sfx_bdcls`, no rebota `p_doors.c:128-157`	`VDOORSPEED*4`

5.3 Constantes de tiempo y velocidad

- **`VDOORSPEED` = `FRACUNIT*3`** `p_spec.h:364`. Con `FRACUNIT = 65536` `m_fixed.h:35-36`, son 3 unidades de mapa por tic para la puerta normal. Las rápidas usan `VDOORSPEED*4`, 12 unidades por tic `p_doors.c:300,325,487,492`. Es lo que hay en este árbol, no la velocidad menor de la build pública. confianza: alta
- **`VDOORWAIT` = `105`** tics `p_spec.h:365`. A `TICRATE = 35` `doomdef.h:122`, 105 ÷ 35 = **3,00 segundos** de espera arriba. No es la espera mayor de la build pública. confianza: alta
- La conversión general es **segundos = tics ÷ 35** `doomdef.h:122`. Por eso la espera de `close30ThenOpen`, `35*30` tics `p_doors.c:144`, son 30 segundos, y la de `P_SpawnDoorRaiseIn5Mins`, `5*60*35` tics `p_doors.c:549`, son 5 minutos.

> **Método de comprobación.** `VDOORWAIT` y `VDOORSPEED` no se contrastan con la Doom Wiki (que puede reflejar otra build). Se comprueban leyendo `p_spec.h` en este árbol y, si se quiere validación dinámica, cronometrando la puerta en el binario compilado desde este código.

6. Vista en ejecución

6.1 El motor de movimiento: `T_MovePlane`

La puerta no mueve el techo ella misma: delega en `T_MovePlane` `p_floor.c:49`, el mismo motor que usan suelos y plataformas. Lo invoca sobre el techo tanto al subir `p_doors.c:170-173` como al bajar `p_doors.c:120-123`, y recibe un resultado: `ok` (se movió sin llegar) `p_floor.c:202`, `pastdest` (alcanzó el destino, el techo se fija exactamente) `p_floor.c:172`, o `crushed` (al bajar pilló algo debajo) `p_floor.c:156-162`. confianza: alta

6.2 Máquina de estados: `T_VerticalDoor`

Thinker que corre una vez por tic mientras la puerta viva `p_doors.c:63`. Ramifica por `door->direction`:

Estado	Qué hace cada tic	Transición
1 subiendo	`T_MovePlane` hacia `topheight` a `speed`	al `pastdest`: `normal`/`blazeRaise` → `0`; `open`/`blazeOpen`/`close30ThenOpen` → se retira `p_doors.c:168-196`
`0` esperando	`if (!--topcountdown)` decrementa	al llegar a 0: `normal`/`blazeRaise` → `-1` con sonido; `close30ThenOpen` → `1` `p_doors.c:69-97`
`-1` bajando	`T_MovePlane` hacia `floorheight`	al `pastdest`: `normal`/`close` se retira; `close30ThenOpen` → `0`. Si `crushed` y no es `close`/`blazeClose` → `1` `p_doors.c:118-166`
2 espera inicial	`if (!--topcountdown)` decrementa	al llegar a 0: solo `raiseIn5Mins` → `1` `p_doors.c:99-116`

Confianza alta para 1/0/-1 (leído en los tres `switch`); media para el `case 2` (solo se ejerce en niveles con el special `raiseIn5Mins`).

6.3 Ciclo de vida de una puerta normal (diagrama de secuencia)

Participantes: `Jugador` `P_UseLines` `P_UseSpecialLine` `EV_VerticalDoor` `T_VerticalDoor` `T_MovePlane`

`Jugador` → `P_UseLines` pulsa «usar» (BT_USE, `p_user.c:322`)

`P_UseLines` → `P_UseSpecialLine` línea especial trazada al frente

`P_UseSpecialLine` → `EV_VerticalDoor` special de puerta (`p_switch.c:338`)

`EV_VerticalDoor` → `EV_VerticalDoor` comprueba cerradura (`p_doors.c:369-410`)

`EV_VerticalDoor` → `T_VerticalDoor` crea `vldoor_t` (dir=1, speed=VDOORSPEED, topwait=VDOORWAIT)

> bucle · cada tic, dir=1 (subiendo)
>
> `T_VerticalDoor` → `T_MovePlane` mueve techo hacia `topheight`; devuelve ok / pastdest

`T_VerticalDoor` → `T_VerticalDoor` pastdest → dir=0, topcountdown=topwait (`p_doors.c:181-183`)

> bucle · cada tic, dir=0 (esperando 105 tics = 3,00 s)
>
> `T_VerticalDoor` → `T_VerticalDoor` `--topcountdown`

`T_VerticalDoor` → `T_VerticalDoor` topcountdown==0 → dir=-1, `sfx_dorcls` (`p_doors.c:81-85`)

> bucle · cada tic, dir=-1 (bajando)
>
> `T_VerticalDoor` → `T_MovePlane` mueve techo hacia `floorheight`; devuelve ok / pastdest / crushed

> **alternativa · según el resultado al bajar**
>
> `T_VerticalDoor` → `T_VerticalDoor` llega al suelo (pastdest) → `P_RemoveThinker`, libera el sector (`p_doors.c:136-140`)
>
> `T_VerticalDoor` → `T_VerticalDoor` algo debajo (crushed) → dir=1, `sfx_doropn`, reabre (`p_doors.c:159-163`)

Confianza del diagrama: alta para `normal`; método: seguir las llamadas en los ficheros citados.

6.4 El sonido, evento a evento

Evento	Sonido	Evidencia
Apertura de puerta normal	`sfx_doropn`	`p_doors.c:337-338,449`
Cierre de puerta normal	`sfx_dorcls`	`p_doors.c:83-84`
Apertura de puerta rápida	`sfx_bdopn`	`p_doors.c:327-328,444`
Cierre de puerta rápida	`sfx_bdcls`	`p_doors.c:77-78,132-133`
Rebote al aplastar	`sfx_doropn`	`p_doors.c:161-162`
Puerta con llave sin la llave	`sfx_oof`	`p_doors.c:379,393,407`

Todos se emiten con `S_StartSound` sobre `door->sector->soundorg`, el origen sonoro del sector. confianza: alta

6.5 Creación de la puerta: las tres funciones de arranque

Función	Responsabilidad
`EV_VerticalDoor` `p_doors.c:353`	Puerta manual sin etiqueta. Comprueba cerradura por color `p_doors.c:369-410` si el sector ya tiene puerta activa la reutiliza e invierte el sentido `p_doors.c:416-437` si no, crea el thinker con `direction=1`, `speed=VDOORSPEED`, `topwait=VDOORWAIT` `p_doors.c:464-466` y fija el `type` según el special `p_doors.c:468-494`.
`EV_DoDoor` `p_doors.c:263`	Puerta remota por etiqueta. Recorre los sectores etiquetados `p_doors.c:275`, omite los que ya tienen puerta `p_doors.c:278-279`, y crea uno por sector fijando `topwait` y `speed` y ajustes por tipo `p_doors.c:291-343`. Devuelve 1 si activó al menos una.
`EV_DoLockedDoor` `p_doors.c:206`	Valida la llave (specials 99/133 azul, 134/135 roja, 136/137 amarilla) `p_doors.c:219-257` y delega en `EV_DoDoor` `p_doors.c:259`.

El `topheight` de apertura se calcula como el techo más bajo del entorno menos `4*FRACUNIT` `p_doors.c:497-498`: la puerta deja un hueco de 4 unidades bajo el dintel. confianza: alta Otras dos funciones crean puertas sin intervención del jugador: `P_SpawnDoorCloseIn30` `p_doors.c:505` (cierre a los 30 s, `topcountdown = 30*35` `p_doors.c:521`) y `P_SpawnDoorRaiseIn5Mins` `p_doors.c:527` (apertura a los 5 min, `topcountdown = 5*60*35` `p_doors.c:549`, único origen de `direction=2`).

9. Decisiones de arquitectura (ADR)

ADR-1. La puerta delega el movimiento en `T_MovePlane` y nunca aplasta.

Estado: vigente en este árbol. **Contexto:** el techo de una puerta y el de un suelo o plataforma se mueven igual, y hay que decidir qué pasa cuando la puerta pilla a alguien debajo. **Decisión:** la puerta no implementa su propio movimiento; reutiliza `T_MovePlane` `p_floor.c:49` y lo llama siempre con `crush = false` `p_doors.c:120-123,170-173`. Cuando el motor detecta un obstáculo debajo con `crush == false`, deshace el movimiento y devuelve `crushed` `p_floor.c:156-162` en vez de aplastar. La respuesta («reabrir») la decide `T_VerticalDoor`, no el motor `p_doors.c:159-163`. **Consecuencias:** una puerta nunca mata por aplastamiento (a diferencia de un techo aplastador, que sí pasa `crush = true`); toda la política de «qué hacer al atrapar a alguien» vive en un único sitio, el thinker de la puerta; y cambiar el motor de movimiento afecta a la vez a puertas, suelos y plataformas (ver riesgos).

11. Riesgos y deuda técnica

11.1 Puntos seguros de cambio, con su riesgo

Quiero cambiar…	Toco…	Radio de impacto y riesgo	Conf.
Espera de la puerta arriba	`p_spec.h:365`	todas las puertas con `topwait=VDOORWAIT`. Riesgo bajo: es un tic contado, no toca geometría.	alta
Velocidad de la puerta normal	`p_spec.h:364`	puerta normal; las rápidas escalan con ella. Riesgo medio: una velocidad muy alta puede saltarse el destino en un tic (el motor lo tolera).	alta
Solo las puertas rápidas	*4 en `p_doors.c`	únicamente las blazing. Riesgo medio: son cuatro sitios (`p_doors.c:300,325,487,492`); tocar uno solo deja tipos incoherentes.	alta
Hueco bajo el techo	`4*FRACUNIT`	altura de apertura de toda puerta. Riesgo alto: seis sitios (`p_doors.c:298,307,324,335,498,547`); un valor malo deja la puerta encajada o sin abrir.	media
Daño de bala	`p_pspr.c:633`	pistola, ametralladora y escopeta, las tres vía `P_GunShot`. Riesgo medio: afecta a más armas de las que parece.	alta
Balas por cargador	`p_inter.c:59`	recogida y munición inicial, con el doble en baby/nightmare. Riesgo bajo; ojo al tope `maxammo[0]=200`.	alta

11.2 Casos límite que el código ya maneja

- **Reutilizar una puerta en marcha.** Si usas una puerta que ya se mueve, no se crea una segunda: se reusa el thinker y se invierte el sentido `p_doors.c:416-437`. Sin esto, dos thinkers pelearían por el mismo techo.
- **Un enemigo no puede cerrarte una puerta encima.** En el reuso, si quien la usa no es el jugador, no se fuerza el cierre `p_doors.c:430-431`.
- **Puertas remotas duplicadas.** `EV_DoDoor` omite los sectores etiquetados que ya tienen puerta activa `p_doors.c:278-279`.
- **Aplastamiento sin muerte.** La puerta pasa `crush = false`: atrapar a alguien debajo devuelve `crushed` y dispara el rebote, no daño `p_floor.c:156-162`.
- **Puertas de un solo uso.** Los tipos `open` por línea anulan su special tras dispararse `p_doors.c:482,491`.
- **Munición al tope.** `P_GiveAmmo` rechaza la recogida si ya estás al máximo `p_inter.c:87-88`.

11.3 Constantes fuera de la puerta relevantes para el examen

No son de la puerta, pero forman parte de las cuatro mutaciones:

- **Daño por bala** (`P_GunShot`): `damage = 7*(P_Random()%3+1)` `p_pspr.c:633` → **7, 14 o 21**, no 5/10/15. Todas las armas de bala pasan por `P_GunShot`: pistola `p_pspr.c:661`, ametralladora `p_pspr.c:753` y escopeta, siete perdigones en bucle `p_pspr.c:686-687`. confianza: alta
- **Cargador de balas** (`am_clip`): `clipammo[NUMAMMO] = {15, 4, 20, 1}` `p_inter.c:59` el primer elemento es **15**, no 10. Tope `maxammo[0] = 200` `p_inter.c:58`. `P_GiveAmmo` multiplica por `clipammo[ammo]` `p_inter.c:91` en baby y nightmare la munición se duplica `p_inter.c:95-98`. confianza: alta

12. Glosario

Tic paso de simulación del motor; 35 por segundo `doomdef.h:122`. Base de toda conversión a tiempo (segundos = tics ÷ 35).

FRACUNIT unidad de coma fija, `65536` `m_fixed.h:35-36` equivale a una unidad de mapa.

Thinker rutina que el motor ejecuta una vez por tic para animar un objeto vivo (aquí, la puerta).

Special número que marca una línea del nivel como puerta, puerta con llave, interruptor o disparador.

Walkover activación por pisar una línea, frente a la activación manual (usar) o por interruptor.

Sector región del nivel cuyo techo o suelo puede moverse; la puerta mueve el techo de su sector.

Incertidumbre declarada (dónde el código no basta)

- El valor absoluto de `topheight` depende de la geometría del nivel (`P_FindLowestCeilingSurrounding`), no de una constante: no se puede afirmar una altura fija sin un WAD concreto. Confianza baja sobre «cuánto sube en unidades absolutas».
- El tiempo total de apertura en segundos depende de la distancia a recorrer (que depende del nivel) dividida por `VDOORSPEED`: la constante fija la velocidad, no la duración total. Solo `VDOORWAIT` es un tiempo fijo y citable en segundos.

- El daño por bala es aleatorio dentro de un conjunto, no un valor único: la afirmación correcta es «uno de 7, 14 o 21», no un número fijo por disparo. Confianza alta sobre el conjunto, no sobre el valor de un disparo concreto.