

idAI2

An entity for AI/NPCs, usually demons.

`idAI2` can be individually placed throughout a map, or more preferably, spawned with an **`idTarget_Spawn`**. The exact ai type of the `idAI2` can be checked by looking at the `inherit` parameter.

Usage

This is an example of a Hell Soldier AI.

`idAI2` entities can greatly vary based on the AI.

```
entity {
  layers {
    'spawn_target_layer'
  }
  entityDef example_ai_soldier_eternal {
    inherit = "ai/fodder/soldier_blaster";
    class = "idAI2";
    expandInheritance = false;
    poolCount = 0;
    poolGranularity = 2;
    networkReplicated = true;
    disableAIPooling = false;
    edit = { // See "struct idAI2 : public idActor"
      highlightDecl = "glorykill_highlight";
      clipModelInfo = {
        type = "CLIPMODEL_BOX";
        size = {
          x = 0.600000024;
          y = 0.600000024;
          z = 1.829;
        }
      }
    }
    dormancy = {
```

```

    delay = 30;
    distance = 19.5070019;
}

spawn_statIncreases = { // Increase these stats on spawn
    hum = 1;
    item[0] = {
        stat = "STAT_AI_SPAWNED";
        increase = 1;
    }
}

targetingDecl = "characters/soldier_blaster"; // Targeting Decl ( defines specific aim assist
/ AR target points )

actorConstants = { // See "struct idActor::idActorConstant"
    perception = {
        eyeOffset = { // offset of eye relative to physics origin {{ units = m }}
            z = 1.71500003;
        }
        crouchedEyeOffset = { // eye offset when crouched {{ units = m }}
            z = 1.06700003;
        }
    }
    actorSounds = {
        sndFootsteps = "footsteps/hellified_soldier/hs_footstep";
        sndRagdollStart = "play_hell_soldier_death_short";
    }
    footstepEffectTable = "impac effect/footsteps/ai_soldier";
    footstepEvents = "footstepevents/default";
    painInfo = {
        decayDelay = 1000; // delay for the decay, in game ticks.
        bucketMaxValue = 400; // the max value for the leaky bucket
        decayRate = -20; // the decay rate for the bucket
    }
    bulletPenetrationData = {
        energyCostToPenetrate = 10; // costs this much penetration energy to penetrate this actor
        damageScaleToPenetrate = 0.75; // bullet damage is scaled by this amount after penetrating
    }
    footstepEffectTable_Sprint = "impac effect/footsteps/ai_soldier_sprint";
    footstepEffectTable_SlowWalk = "impac effect/footsteps/ai_soldier";
    footstepEffectTable_CrouchWalk = "impac effect/footsteps/ai_soldier";
    footstepEffectTable_Landing = "impac effect/footsteps/ai_soldier_landing";
}

```

```

[[[footstepEffectTable_HeavyLanding = "impacteffect/footsteps/ai_soldier_landing";
[[[ledgeGrabEffectTable = "impacteffect/footsteps/ai_soldier";
[[[ledgeGrabEffectTable_Heavy = "impacteffect/footsteps/ai_soldier";
[[[ledgeGrabEffectTable_Friendly = "impacteffect/footsteps/ai_soldier";
[[[ledgeGrabEffectTable_FriendlyHeavy = "impacteffect/footsteps/ai_soldier";
]]]
[[actorEditable = { // See "struct idActor::idActorEditable"
[[[entityDamageComponent = { // names of joint groups, their associated damage scale, and armor
level
[[[entityDamage = "entitydamage/ai/soldier_blaster/base";
]]]
[[[injuredStates = { // defines parameters of each injured state, see "struct idInjuredState"
[[[hum = 1;
[[[item[ 0] = {
[[[[name = "not_injured";
[[[[damageGroupMaxGoreLevels = {
[[[[hum = 6;
[[[[item[ 0] = {
[[[[[damageGroupName = "head";
[[[[[maxGoreLevel = "GORELEVEL_TATTERED";
[[[[]]}
[[[[item[ 1] = {
[[[[[damageGroupName = "chest";
[[[[[maxGoreLevel = "GORELEVEL_TATTERED";
[[[[]]}
[[[[item[ 2] = {
[[[[[damageGroupName = "right_arm";
[[[[[maxGoreLevel = "GORELEVEL_TATTERED";
[[[[]]}
[[[[item[ 3] = {
[[[[[damageGroupName = "left_arm";
[[[[[maxGoreLevel = "GORELEVEL_TATTERED";
[[[[]]}
[[[[item[ 4] = {
[[[[[damageGroupName = "right_leg";
[[[[[maxGoreLevel = "GORELEVEL_TATTERED";
[[[[]]}
[[[[item[ 5] = {
[[[[[damageGroupName = "left_leg";
[[[[[maxGoreLevel = "GORELEVEL_TATTERED";

```

```

    }
}

allowIK = true; // true if AI uses IK in this injured state
canUseAllTraversalsWhileInjured = true; // true if AI can use all traversals in this injured state
canUseDownTraversalsWhileInjured = true; // true if AI can use down traversals in this injured state
}

}

radiusDamageJoints = { // names of joints to trace to for radius damage tests
    num = 6;
    item[0] = "head_part01_md";
    item[1] = "spine_part01_md";
    item[2] = "arm_hand_lf";
    item[3] = "arm_hand_rt";
    item[4] = "leg_lower_lf";
    item[5] = "leg_lower_rt";
}

}

factionName = "blaster";
mass = 18.1439991; // mass of the actor {{ units = kg }}
lootable = false;
lootDropComponent = { // lootdrop decl for campaign
    lootDropDataDecl = "ai/default_fodder";
}

pvpLootDropComponent = { // lootdrop decl for Battlemode
    lootDropDataDecl = "ai/default_fodder_pvp";
}

aiConstants = { // See "struct idAI2::idAIConstant"
    components = {
        ptr = {
            ptr[12] = {
                componentDecl = "aicomponent/pathmanager/base";
            }
            ptr[14] = {
                componentDecl = "aicomponent/attack/base";
            }
            ptr[9] = {
                componentDecl = "aicomponent/positionawareness/soldier_blaster/base";
            }
        }
    }
}

```

```

[ptr[10] = {
[componentDecl = "aicomponent/extendedsense/soldier_blaster/base";
}
[ptr[11] = {
[componentDecl = "aicomponent/transientfocus/soldier_blaster/base";
}
[ptr[13] = {
[componentDecl = "aicomponent/soldier_blaster";
}
}
}
[syncMelee = { // See "struct idSyncAttack"
[msAfterAttackBeforeCanSync = 250; // how long after a regular attack before a sync attack can
be performed
[syncMeleeEntityDefs = { // entityDef decls for syncmelee
[hum = 2;
[item[0] = "syncmelee/soldier_blaster";
[item[1] = "syncmelee/soldier_blaster_3p";
}
[syncGroups = { // See "struct idSyncAttack"
[hum = 1;
[item[0] = {
[syncGroupName = "";
[syncInteractions = { // syncInteraction decls
[hum = 19;
[item[0] = "syncdeath/playersai/soldier_blaster/right_upper";
[item[1] = "syncdeath/playersai/soldier_blaster/left_upper";
[item[2] = "syncdeath/playersai/soldier_blaster/front_upper";
[item[3] = "syncdeath/playersai/soldier_blaster/front_head";
[item[4] = "syncdeath/playersai/soldier_blaster/left_lower";
[item[5] = "syncdeath/playersai/soldier_blaster/above_back";
[item[6] = "syncdeath/playersai/soldier_blaster/above_front";
[item[7] = "syncdeath/playersai/soldier_blaster/front_rightarm";
[item[8] = "syncdeath/playersai/soldier_blaster/front_leftarm";
[item[9] = "syncdeath/playersai/soldier_blaster/berserk/berserk_above_front";
[item[10] = "syncdeath/playersai/soldier_blaster/chainsaw/cut_back";
[item[11] = "syncdeath/playersai/soldier_blaster/back_lower";
[item[12] = "syncdeath/playersai/soldier_blaster/chainsaw/cut_front";
[item[13] = "syncdeath/playersai/soldier_blaster/berserk/berserk_front_upper";
[item[14] = "syncdeath/playersai/soldier_blaster/right_lower";

```

```

        item[15] = "syncdeath/playersai/soldier_blaster/back_upper";
        item[16] = "syncdeath/playersai/soldier_blaster/crucible/crucible_front";
        item[17] = "syncdeath/playersai/soldier_blaster/crucible/crucible_back";
        item[18] = "syncdeath/playersai/soldier_blaster/front_lower";
    }
}

alwaysAllowChainsawGloryKill = true; // if true then always allow chainsaw glory kills
without needing stagger states or other such things

}

aiDeathStat = "STAT_HELL_MARINE_KILLED"; // what stat to increase on death of this ai
positioningParms = { // parms used to control major positioning in behaviour finite state
machine
    num = 2;
    item[0] = "soldier_blaster/plasma";
    item[1] = "soldier_blaster/plasma_object";
}
aiDeathCodex = "codex/hell/demon_soldier_blaster"; // what codex to give the player when this
ai dies
}

aiEditable = { // See "struct idAIEditable"
    perception = { // See "struct idAIPerception"
        actorPerceptionRadius = 39; // max radius at which AI can perceive other actors {{ units = m
        }}
        obstaclePerceptionRadius = 78; // max radius at which AI can perceive obstacles {{ units = m
        }}
        closePerceptionRadius = 5; // max radius .... for close FOV {{ units = m }}
        eventPerceptionRadius = 39; // max radius at which AI will perceive any events {{ units = m
        }}
        senseUpdatesOnNonEnemies = false; // if true, AI will skip all worldstate updates for non-
        enemies
    }
    useTouchComponent = true;
    death = { // See "struct idAIEditable::idAIDeath"
        ignoreDamageType = "DAMAGETYPE_EMP"; // entity will not take any damage from any damage decl
        that ANDs with this value
        fadeOutAfterDeathDelay_Seconds = { // delay burn-away fx until at least after this much time
        ( 0 uses default, negative implies never )
        value = 3;
    }
}

```

```

    removeAfterFadeOutDelay_Seconds = { // remove the entity this long after burn away fx start
        value = 3;
    }

    canBecomeInjured = false; // if true, AI can become injured and use injured runs and idles
    explosionDecl = "ai/default";
    declTwitchPain = "twitchpain/soldier_blaster";
    deathAnim = ""; // animation to force on death
    trigger = ""; // idEntity to trigger on death
    triggerGloryKill = ""; // idEntity to trigger on death by glory kill
    triggerNonGloryKill = ""; // idEntity to trigger on death by normal kill
    onlyDieByGK = false; // true if the AI will only die as a result of receiving sync damage
    (glory killed)
}

movement = { // See "struct idAIEditable::idAIMovement"
    wanderRadius = 19.5070019; // how far an AI can randomly wander from its spawn position {{
    units = m }}
    useTraversalClassA = true; // if true, can use class A traversals
}

cover = { // See "struct idAIEditable::idAICoverInfo"
    coverRadius = 19.5070019; // max radius at which the AI will check for cover {{ units = m }}
}

behaviors = { // See "struct idAIBehaviors"
    decl = "behaviors/soldier_blaster/soldier_blaster"; // the behavior typeinfo decl this
    behavior uses.
    declBehaviorEvents = "behaviorevents/default"; // AI events
    attackGraph = "ai/soldier_blaster"; // available attacks
}

vsAIDamageMask = "HEALTH"; // damageCategoryMask_t; how to process damage taken from other AI
spawnSettings = { // Refer to "struct idAIEditable::idAISpawn"
    entranceAnimPath = "animweb/characters/monsters/soldier_blaster/spawn/teleport_entrance"; //
    initial animation to play specified via animref if entranceAnim not given
    spawnFXEntityDef = "fx/spawn_in_fodder"; // entity def for fx to use when spawned with
    AIOVERRIDE_TELEPORT
    initialState = "AIOVERRIDE_TELEPORT"; // initial value for aiStateOverride_t; can be
    overwritten in the spawn target
    teleportDelayMS = 750; // How long it takes to spawn in when using AIOVERRIDE_TELEPORT
    chanceMissingArmor = { // See "struct idAIEditable::idAIMissingArmor"
        num = 0;
        item[ 0] = {
            damageGroup = "head"; // damage group whose armour may or may not be there

```

```

        missingChance = 100; // 0-100 % chance that armour is missing
    }
}

demonTeamInfo = {
    canHostLostSouls = true;
}

staggerEnabled = true; // Set to false to disable staggering
}

aiHealth = { // Refer to "struct idHealthT < aiHealthComponent_t , AI_HEALTH_MAX ,
AI_HEALTH_HITPOINTS >"
    components = {
        components[1] = {
            max = 0; // Max regen allowed
            regenInterval = { // interval, in seconds, between regen updates.
                value = 0;
            }
        }
        components[0] = {
            max = 400; // Max HP allowed
            starting = 400; // HP when spawned
        }
    }

    goreComponent = { // See "struct idGoreComponent"
        goreContainer = "ai/fodder/soldier"; // gore component decl
    }

    afProperties = { // See "struct idAnimator_AF : public idAnimator_Base"
        impactEffectTable = "impacteffect/ragdoll/ragdoll_fodder"; // impact table for sound and
particles
        articulatedFigure = "characters/monsters/soldier_blaster_auto"; // the articulated figure
decl to use
    }

    renderModelInfo = {
        model = "md6def/characters/monsters/soldier_blaster/base/soldier_blaster.md6"; // md6def decl
to use
        lightRigDecl = "soldier_blaster/soldier_blaster_default";
        materialRemap = { // Use to replace the textures on a per entity basis
            num = 0;
        }
    }
}

```

```

    }
    fxDecl = "character/hellified_soldier_blaster/hellified_soldier_blaster"; // fx decl for this
entity
    startingInventory = { // See "struct idInventoryAttachmentDef"
        num = 2;
        item[0] = {
            startSlot = "EQUIPPED"; // startingSlot_t; EQUIPPED, HOLSTERED, or BACKPACK
            inventoryDecl = "weapon/ai/soldier_blaster/plasma"; // inventory decl for this attachment
        }
        item[1] = {
            showHolstered = false; // if true, attach and show the item when it's holstered
            inventoryDecl = "weapon/ai/soldier_blaster/plasma_slug";
        }
    }
    walkIKDecl = "walkik/biped_base";
    killerNames = { // What to string display when killed by this entity
        num = 1;
        item[0] = "#str_decl_demoncard_summon_fodder_blaster_soldier_name_GHOST53375";
    }
    spawnPosition = { // Spawn position does not really matter
        x = 1;
        y = 1;
        z = 1;
    }
    flags = {
        hide = true;
    }
}
}

```

Revision #20

Created 8 September 2021 19:42:17 by Kaiser_will

Updated 24 August 2022 18:43:55 by Velser