

Contents

1. Path placement system	3
1.1. Detailed Description of Path.....	4
2. Stalker's behavior schemes.....	7
2.1. walker Scheme.....	7
2.2. camper Scheme.....	7
2.2.1.sniper Scheme.....	8
2.3. remark Scheme.....	9
2.4. patrol Scheme.....	9
2.5. sleeper Scheme.....	10
2.6. kamp Scheme.....	10
2.7. wounded Scheme.....	10
2.8. Additional Scheme internal settings.....	12
3. Configuration Logic by Event.....	12
3.1. Section combat.....	12
3.2. Section death.....	12
3.3. Section hit.....	12
3.4. Section danger.....	13
3.5. Additional Sections	13
3.5.1. Section dont spawn character supplies.....	13
3.5.2. Section dont spawn loot	14
3.5.3. Section spawner.....	14
3.5.4. Section known info.....	14
4. Monsters logic schemes.....	14
4.1. Scheme mob walker	14
4.2. Scheme mob remark.....	15
4.3. Scheme mob combat, mob_death.....	15
4.4. Scheme mob_jump (spring-monster).....	15
4.5. Scheme mob home.....	16
4.6. Additional internal scheme settings for monsters.....	16
5. Logic settings and switching between schemes.....	16
5.1. Settings inside section.....	18
5.1. Setting conditions and running functions.....	18
5.2. Working with sound.....	19
5.4. Counters.....	19
5.3. Cutscenes.....	20
5.5. Postprocessings.....	20
5.6. Digital identifiers, story id.....	21
5.7. Logic examples.....	21
6. Logic schemes space restrictor.....	24
6.1. Scheme sr_idle.....	24
6.2. Section sr_no_weapon.....	25
6.3. Section sr_light.....	25
6.4. Scheme sr_particle.....	25
6.5. Scheme sr_timer.....	26
6.6. Scheme sr_psy_antenna.....	26
6.7. Scheme sr_cutscene.....	27
7. Additional logic settings for other objects.....	27
7.1. Door working scheme, section ph_door.....	28
7.2. Button working scheme, section ph_button.....	28
7.3. Gate working scheme, section ph_gate:.....	29

7.4. Code locks, ph_code:	30
7.5. Push physics object, ph_force:	30
7.6. Object throwing denying, ph_heavy:	31
7.7. Hanging Physics, ph_oscillate:	31
7.8. Reaction on sound:	31
8. Controlling some features:	32
8.1. NPC reaction settings, meet manager:	32
8.2. Smart covers (cs-cop):	35
8.3. Text outputting:	35
8.4. Activation\Deactivation Anomalies:	36
8.5. Working with text:	36
8.6. Disabling post combat:	36
8.6. Physics objects settings:	37
9. LTX files syntaxes:	38
9.1. Sections, keys:	39
9.2. Inheritance:	40
9.3. Includes:	41

1. Path placement system

At the waypoints you can specify flags that modify the behavior of the character. Flags are sets directly in the name of waypoint, example for the point named, "wp00":

wp00|flag1|flag2

Waypoints flags path_walk:
a=state

Selects the state of the body when you move (only the section-walking state) List of states can be taken in gamedata\scripts\state_lib.script

p=percent

The probability of staying at the point in percentage (0 - 100). Default is 100, means that stalker never goes past a stop points.

sig = name

Set the signal with the "name" immediately upon arrival at the point (before turning around) for its subsequent verification by the field on_signal logic. If you want to set a signal after turning - use the waypoint flag path_look.

Flags waypoints path_look:

a = state

Selects the state of the body when standing (or sitting) on the point. (From Section Stand-up and Sedentary state) List of states can be taken at gamedata \ scripts \ state_lib.script

t = msec - Time in milliseconds, that a character should look at a given point.

'*' - An infinite time. Valid values in the range [1000, 30000], default - 5000. For the final (terminal) vertices of the path path_walk, in which no more than 1 second corresponding point path_look, the value of 't' is always assumed to be infinitely and obviously do not need to be set.

sig = name

After turning at the point path_look, set an signal with name name.

‘syn’

A check mark will delay the installation of the signal as long as the point with a flag syn does not come with all the characters with the team (team is defined as a text string in customdata). Until then, while the other characters do not come expecting the character will regain its idle animation.

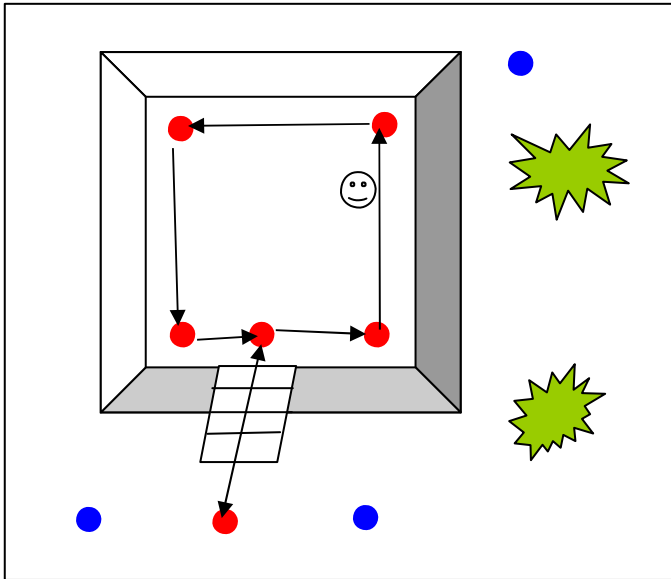
`sigtm = signal`

Sets the signal when you call `time_callback`-and state manager-ohm. Accordingly, if $t = 0$, then the signal will be installed after winning back init animation. It is used, for example, animation press, which consists of two parts: 1 - click on the button, 2 - omit hand. In a way `path_look` can do: `wp00 | a = press | t = 0 | sigtm = pressed`. And then switch scheme: `on_signal = pressed | other_sheme`

1.1. Detailed description of path.

Walker.

Configuring



At the map for each walker you need to set:

- 1) Path **path_walk**, on which walker walks.
- 2) The Way **path_look**, consisting of the points in which the walker looks.

Walker-s may be 1 or more. They can act independently or interact with each other.

[Walker]

team = ...

command name, an arbitrary text string. All walker-s on the same team must have the same "team". It is advisable to specify the name of the team level and the name of the place where there are walker-s, for example: escape_bridge, escape_factory, it will reduce the chance of mistakes and give a different command common name.

path_walk = ...

pathname, as described in Section 1 path_look = ...

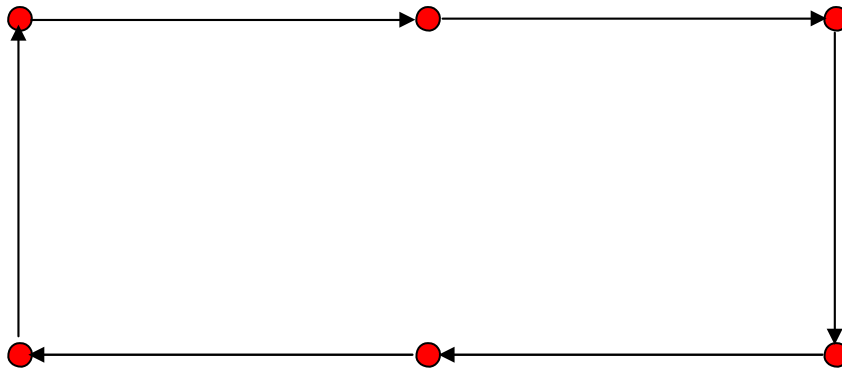
(Optional) path name described in Section 2. If the character is just to walk along the route, path_look can not be set.

If a character needs to stand still, he is given one waypoint path_walk and at least one waypoint path_look

Rules for placement of flags in the ways we consider a few examples:

Example 1:

Character patrolling the area around the two houses. The route is constructed as follows:



How to make a character running or crouching between certain points ? To do this there are flags at the way path_walk

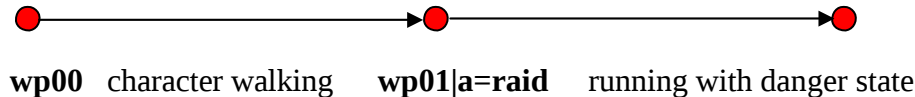
Each has a name of waypoint: wp00, wp01, etc.

Flags are sets in that name. They need to be separated from the name with the character '|'.

like this: a = anim, where anim - title animation from paragraph 2.4.4. of this documentation. If we write a = threat then the character will walk at Danger state if a = raid then flee with the weapon at the ready, etc.

NB: At way points path_walk animation used ONLY from the section "walking state"!

Example 2:



Character Talking

To make able the character talking, while moving along the route, you need to determine at each point a list of topics on which he can speak. To do this, there are the following fields:
s = name_sound_scheme (default is mute). Several themes can be listed separated by commas.

Example 3:

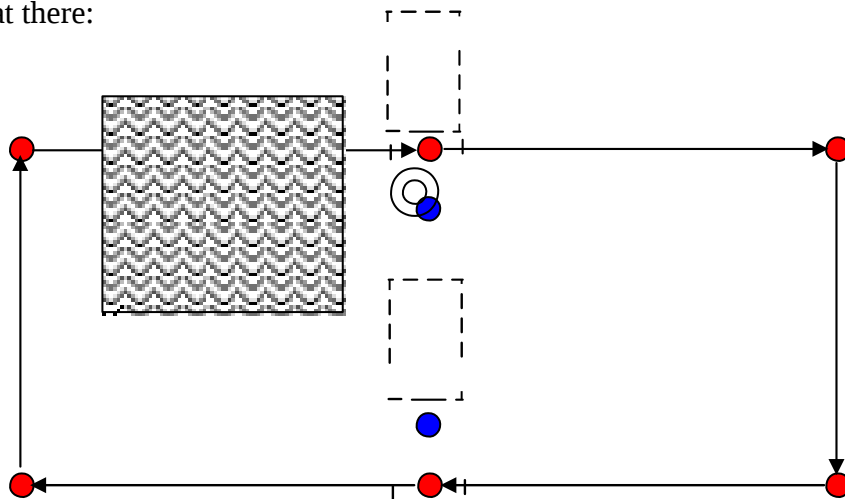


In Example 3, using only the field of s, to specify the topic of conversation, and flag sc, to show that not a single sound is played, but from time to time.

Setting other parameters (sp, sf, st) NOT RECOMMENDED, default is acceptable for most scripts. Parameter sa is also not recommended. If you want to start a sound simultaneously with the animation, the best way to use fields points path_look, which will be written later in this document.

If the character is not just walking along the route, but must also stop and play the animation, you need to set him the way `path_look`.

Example 4: perfecting the example 1, so that the character, past the opening between the houses, stopped and looked at there:



What's new in this example? `Path_look` contain two points. Connection between the points of this path is recommended immediately to remove with the editor, because its not used.

Next, at the waypoints `path_walk` `path_look`, which are encircled by dashed line, you need to put the similar flags in the editor. For example, the top pair of points put flag 0, while the bottom pair of points - flag 1.

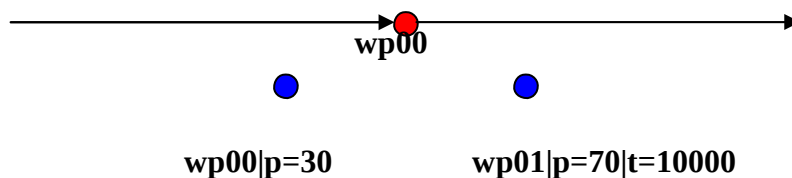
Now, the character will stop at points `path_walk`, flagged, and look to the point `path_look`, marked by the same flag. If the point `path_walk` not flagged, character is not stopping. One point `path_walk` may correspond to several points `path_look`. Then the character choosees randomly one of the matching points.

By analogy with `path_walk`, at waypoints `path_look` you can use different flags that change the behavior:

$p = 100$ - the likelihood that the character will look specifically at this point. P values of all matching points are summed, so if one of $p = 100$, while the other 300, then the character will look first to a 25% chance! (100 out of 400). To avoid confusion, it is recommended to set p so that their sum is 100. By default, all points $p = 100$.

t = time at which a character is delayed at this point (default is 5000 milliseconds)

Example 5:



In this example, passing through the point `wp00`, a character with a 30% chance look at the point `wp00` within 5 seconds, but with a probability of 70% will look into the point `wp01` for 10 seconds.

By default, when stopped, character playing animation idle, if he is unable to crouch, or animation hide, if he is able to crouch.

If you want a different animation, you can specify it with a flag:

a = animation_name (default idle).

means, a = anim, where anim - title of animation from paragraph 2.4.4. of this documentation. If we write a = hide, then the person sits in a Danger state, if a = guard, then get up with guns at the ready, etc.

NB: At the way points path_look you can use animations ONLY from the section "Stand-up and sit-state"!

2. Stalker's behavior schemes.

There is a certain set of schemes that describe the behavior of the character. They are written in his custom_data or, in the case of Smart, the relevant files describing the working operation of Smart. Below is a list of these schemes.

2.1. Scheme walker

This basic scheme, in which the character moves on patrol path (path_walk) and stops at certain points and performs the appropriate action.

* - optional

[walker]
path_walk = <path name> - main NPC's walking route

*path_look = <path name> - the path for NPC looking. At points path_walk, which correspond to point the way path_look (are the same flags), the character stops and looks at some point.

Not mandatory, only if the path path_walk more than one point of the path

*team = <team_name> - synchronization team

* def_state_moving1 = - State (animation), in which stalker moves to the first point of the way, if it is close (patrol default)

* def_state_moving2 = - State, in which stalker moves to the first point of the way, if its not so far (*rush* by default)

* def_state_moving3 = - State, in which stalker moves to the first point of the way, if its far away from him (*sprint* by default)

* def_state_standing = - default state where he stands and looks at the point, if at this point is not specified other state (*guard* by default).

The scheme reads the standard signal:

path_end - NPC reached the endpoint of the path

2.2. Scheme camper

camper properties:

- Camper stands at the point and looking in the direction of where you placed it in the editor or moving by the patrol ways
- Campers are switched to a universal combat only when they see the enemy is closer than 30 meters. If he survived, returns to camper status.
- In all other cases, act on their own scripting scheme. If you see an enemy - Shoot. If you hear danger - then look in the direction of Danger. If you see a grenade - running away from grenades. If you have seen the enemy and the enemy was gone, then look at the point where last seen the enemy.
- Campers are not fighting on the move. If they see the enemy - they stop, shoot, and then continue to move.

[camper]

path_walk = *patrol_path*

**path_look* = *patrol_path* - optional, only if the route *path_walk* contains more than one point of the path

**radius* = - distance in meters, if the distance between Camper and the enemy is less than specified, Camper goes to the universal combat. By default, this radius is 20 meters.

**no_retreat* = *true* - the character at the sight of the enemy will not run at the closest point *path_walk*, but immediately returns to the killing. Need it if you want to do a scene where some guys bump on the others. Put Campers with the above flag. They will go on their patrol routes and will killing enemies.

**def_state_moving* = - Animation, in which the stalker moves to the nearest point of the path when the enemy is close (default *sneak*)

**def_state_moving_fire* = - State (animation), where stalker shoots back from the enemy, while moving to the nearest point of the path (*sneak_fire*)

**def_state_campering* = - state in which stalker awaits the enemy on the path (*hide*)

**def_state_campering_fire* = - state in which stalker shoots back from the enemy, while on the path (*hide_fire*)

**attack_sound* = name of sound theme

Opportunity to override sound of the attack for the snipers / campers. By default it is equal to sound the theme "fight_attack". You can change to any other (for stage requirements) or even disable the prescribing section
Camper: *attack_sound* =

**shoot* =

Type of shooting options:

always

- (default) always shooting when possible

none

- never shoots.

terminal

- shoots only when it is at the last points of patrol route

(This is done to ease the construction of attacking scenes).

ATTENTION! Campers have one big disadvantage - when he takes a hit and he does not know from where the hit is applied (can not see the enemy or does not hear the shot), he stupidly continues to stand at the old place and wait for the next bullet.

Because of this it is not necessary to arrange campers when stalkers have to defend themselves and hold a position in the event that there are several areas where the player or stalkers can attack Camper. Use walkers in such cases.

Campers are good to be put on attack at the routes and as snipers.

Standart scheme signal:

path_end – NPC finished the path

2.2.1.Scheme sniper

Variety of Camper, firing only with single shots and not looking at the points of the patrol path but scanning space between them. The scanning speed from point to point fixed and equal to 20 seconds.

NB! Set only 2 look points for sniper!

Edit the Camper section:

```
[camper]
path_walk = walk1
path_look = look1
sniper = true
```

2.3.Scheme remark

Scheme is used for synchronization a bunch of other schemes, or playing the animations and replies . Scheme remark can be used, without specifying the parameters in this case will take the default settings.

```
[remark]
anim = remark animation,by default wait
target = Stalker looking direction
```

```
target = story | actor or story_id      – look at the player or object with specified
story_id
target = path | patrol_path, point_id  – look at the patrol path, where patrol_path –
name of way, and point_id – number of way point
target = job | job_section, smart_name – look at the job inside specified в Smart,
where job_section – name of job, and smart_name – name of smart terrain.
target = nil                           – look at nowhere.
```

Example:

target = vasya – character will looking at the object with story_id - vasya

Standart signals for remark:

```
sound_end      – after ending of sound scheme
anim_end       – after anim playing
action_end     – after playing both of them, if they are synchronized
```

Example of sound and animation synchronization with remark scheme:

```
[remark]
anim = animation snd
=          sound
snd_anim_sync = true
on_signal = action_end | next scheme
```

Field anim_resetin scheme [remark] not working.

2.4. Scheme patrol

For creation of patrol (Variation of kamp scheme only with walking state)

[patrol]

path_walk = *path_walk*

* *path_look* = *path_look* - optional, only if the route *path_walk* contains more than one point of the path

* *formation* = *back*

* *commander* = *true* – set the commander (the one handsome man :))

* *move_type* = *patrol* - sets the initial moving mode, by default patrol.

Animations listed at file *state_mgr_lib*

* *formation* = *back* – formation mode (not necessary)

back - a little backward from commander in two lines (by default)

line -line formation

around - around commander

When commander stops to "**meet**" the commander's men stops as well.

If the commander dies, it will automatically select another. Commander is the one who first came under the scheme. Methods of construction can be given at waypoints follows: *ret* = 0 ... 2

0 – line

1 – around

2 – back

When moving, Commander works like a normal walker and the accompanying mobs repeat his actions. So, if the parameters of waypoint set to *a* = assault, then he will rush off with his weapon, and the rest of his squad will copy this action.

2.5. Scheme sleeper

Scheme of sitting and sleeping NPC. Need to put a patrol road, a minimum of 1 point. Sleeper will sit down to sleep in the zero point of the path and turn into direction of first point.

[sleeper]

path_main = <*path name*>

* *wakeable* = *true* – can fast wakeup (if true, then sleeps on his heels and mumbling while he is sleeps)

NB: If the path consists of two points, then the connection needs to be done from the first point to zero (or bidirectional).

2.6. Scheme kamp

Scheme of stalker, sitting in a certain radius around a specified point (the fire), and facing at this point.

[kamp]

center_point = *kamp_center* – name of point around which the NPC will sit.

radius = 2 - how far the stalker will sit from the center of the camp, 2 - default

def_state_moving = *run* – default state in which stalker will go to the kamp point

NB! If the kamp point is in the fire, then offline stalkers comes at her, and when they go into online, they will be inside the fireplace, where they will hit. To prevent this from happening in the kamp section you need to specify *path_walk* from a single point, whose name = *<path_kamp_name>_task* (not need if kamp location in the forest,field,hill etc. (without fireplace)

path_walk = *<path_kamp_name>_task*

2.7. Scheme wounded

wounded configuration

[logic]

active = *walker*

[walker]

wounded = *wounded*

[wounded]

hp_state = *HP|anim@sound|HP|anim@sound*

hp_state_see = *HP|anim@sound|HP|anim@sound*

psy_state = *PSY|anim@sound|PSY|anim@sound*

hp_victim = *HP|nil|HP|actor*

hp_cover = *HP|true|HP|false*

hp_fight = *HP|true|HP|false*

help_dialog = *story_id*

help_start_dialog = *story_id*

Description:

hp_state – behavior when character dont see the player

hp_state_see – when see the player

psy_state – when psy attacks performs on character

hp_victim – where to look, depends on HP level. possible values: ***nil***, ***actor***, ***story_id***

hp_cover – get to the cover or not, depends on HP

hp_fight – can fight or no ,depends on HP

help_dialog – dialog identeficator (instead of standart,if you need special dialog for the story)

Also, we put the starting dialogue of *wounded*. If we put this, all the actors' dialogue for the wounded must have a precondition: *dialogs.allow_wounded_dialog*.

Where:

HP – character health threshold(from 0 to 1)

PSY – psy health threshold

Example with default settings:

[wounded]

hp_state = *30|help_me@help|10|wounded_heavy@help_heavy*

```

hp_state_see = 30|wounded@help_see|10|wounded_heavy@help_heavy
psy_state = 50|{=best_pistol} psy_armed,psy_pain@wounded_psy|20|
           {=best_pistol}psy_shoot,psy_pain@{=best_pistol}wounded_psy_shoot,wounded_psy
hp_victim = 30|actor|10|nil
hp_cover = 30|true|10|false
hp_fight = 30|true|10|false
syndata = wounded@help

```

Where:

best_pistol – Checking that the best weapon of NPC is a pistol.

Example 2 (NPC will not fall as wounded)

```

[wounded]
hp_state = 0|wounded_heavy@help_heavy
hp_state_see = 0|wounded_heavy@help_heavy
hp_victim = 0|nil
hp_fight = 0|false
hp_cover = 0|false

```

2.8. Scheme additional internal settings

```

combat_ignore = true (NPC will ignore the combat)
combat_ignore_cond = {+info -info =func !func} true (NPC will ignore combat by special cond list )
                    condlist combat ignore functions:
                    fighting_dist_ge(distance in meters ) – for
                    combat_ignore,
                    is_enemy_actor – Current enemy is an actor?
                    enemy_in_zone(zone_name) – enemy is in zone
                    (zone_name)
combat_ignore_keep_when_attacked = true (NPC keeps ignore the combat even the actor hits him)
out_restr=<restricor_name> (NPC will not leave restrictor) in_restr =
<restricor_name> (NPC dont enter restrictor)
invulnerable = true (god mode for the character)
show_spot = {+info1} false (NPC radar spot disabeling)

```

3. Logic settings by event.

3.1. Section combat

Shows, what's happening , when NPC goes into the combat.
for modelling script fights uses
combat_type.

```

[logic]
active = walker
on_combat = combat (starts when NPC goes into combat)

```

```
[combat]
on_info = %+info -info =func% round effects
```

```
[walker]
combat_type = {+info =func} camper (monolith, zombied)
path_walk = ...
```

3.2. Section death

NPC dying
on_death = death

```
[death]
on_info = %+info -info =func%
```

3.3. Section hit

NPC takes a hit
on_hit = hit

```
[hit]
on_info = %+info -info =func%
```

3.4. Section danger

You can set this parameters only inside schemes, for example:

```
[walker]
danger = danger_condition
```

```
[danger_condition]
ignore_distance = 50 (distance in meters)
ignore_distance_grenade =
ignore_distance_corpse =
ignore_distance_hit =
ignore_distance_sound =
```

Danger inertion time (default settings)

```
danger_inertion_time_grenade = 20000
danger_inertion_time_corpse = 10000
danger_inertion_time_hit = 60000
danger_inertion_time_sound = 15000
```

The algorithm works as follows: First, verify that the distance to the danger is not clipped by *ignore_danger*. If the danger is closer, then examines its type, and verified by an appropriate distance of this type. If the danger is closer - then allowed the reaction to it.

NB: if necessary that in different cases stalker ignored Danger of different types, you need to create several sections of Danger:

```
[danger_condition@1]
```

ignore_distance = 50

[danger_condition@2]
ignore_distance = 20

* danger_expiration_time = Danger expiration. default 5000 ms.

* danger_inertion_time = how much time needs for NPC to forget the danger default 10000 ms.

3.5. Additional sections

3.5.1. Section dont_spawn_character_supplies

don't spawn supplies to the NPC

[dont_spawn_character_supplies]

3.5.2. Section dont_spawn_loot

loot disabling for quest NPC

[dont_spawn_loot]

3.5.3. Section spawner

This section, which is present in NPC and monsters spawn them on certain conditions (send online). In order that they appear at this point, they should be configured in Level editor and disabled flag no_move_in_offline can_switch_offline. Spawner prescribed at the custom date, before the object logic.

Spawner runs as follows:

[spawner]
cond = {+info -info =func !func}

Note. If the conditions spawn will not run, then the object will not be spawned, and if condition ceases to be valid, the object will be led away in offline by spawner.

Example:

[spawner]
cond = {=is_day}

3.5.4. Section known_info

after NPC's body looting, gives you info porshn

[known_info]
info1
info2

4. Monsters logic shemes

4.1. Scheme mob_walker

most the same as walker. but with few differences.

Moving path flags:

s=sound_scheme (idle, eat, attack, attack_hit, take_damage, die, threaten, steal, panic, growling)

c=true - crouching; r=true - run forward sig=signal_name - set

signal for xr_logic

Observation path flags:

t=time_msec - time in ms for waiting and looking at the point

a=animation (attack, capture_prepare, danger, eat, free, lie_idle, look_around, panic, rest, sit_idle, sleep, stand_idle, turn)

in customdata of character set (*necessary fields):

[walker]

path_walk = путь перемещения

path_look = путь обзора

*no_reset = true/false - dont remove previous scheme (if you need to save sound for example). Default false.

*actor_friendly = true/false never attacks until player hit mutant , default false.

*npc_friendly = true/false - same to other mutants,default false

*friendly = true/false - not attacks player or other mutants,default false.

File: \gamedata\scripts\mob_walker.script

Bloodsucker invisibility:

[mob_walker]

...

state = vis

or

state = invis

Sets default value.

Inside flags walk of path mob_walker you also can use flag b (behaviour) with same parameters:

wp00|b=vis

wp00|b=invis

4.2. Scheme mob_remark

*state = special mutant state (invisibility for bloodsuckers for example)

*dialog_cond = {+info, =func, -info, !func} dialog window condition

*anim = monster animations name,name2,name3 etc.

*anim.head = head anims

*tip = cursor

**snd* = sound of mutant
**time* = animation playing time (debugging only).

4.3. Scheme **mob_combat**, **mob_death**

Same as Stalkers.

4.4. Scheme **mob_jump**

Jumping parameter without restrictions

Example

```
[logic]
active = mob_jump

[mob_jump]
path_jump = path
ph_jump_factor = 2.8
offset = 0,10,0
on_signal = jumped | nil
```

path_jump - the path by which we define a target point jump (with index zero). The real point takes into account the position of path_jump [0] + offset specified by offset.

offset - offset of the axes x, y, z, respectively, that specifies the real point in space (maybe not in the au-node).

ph_jump_factor - affects the time jump. Visually, it is given by the curvature of the trajectory. What it is, the sharper the jump, a quick (less than an arc). With this scheme you can do: jumping off a building to a building, jumping out of windows, jumping high fences, etc. default value = 1.8

Note:

In fact mob_jump - is not a state, its single action. Monster turns toward the jump and jumping, picking up the signal jumped. like "on_signal = jumped | schme_name_or_nil"- is a required parameter in the scheme to know where to go next.

When you select a position using the first point of the patrol path (0 index)

4.5. Scheme **mob_home**

Monsters will walking or sleeping around this point

Example:

```
[mob_home]
path_home = path1
home_min_radius = 10
home_max_radius = 30
aggressive_home - mutants running (not walking) at point path_home
```

Description:

Monsters are held around the way points path_home. In the rush to attack the enemy, if the enemy inside the radius home_min, otherways - hide in covers. It follows that home_min-radius ,and you need to make inside it enough covers. In Idle they also usually disagree on covers. Home_max radius is made on the basis of a large restrictor in the scheme of "nest".

Added ability to set minimum and maximum radius for the scheme mob_home in flags of the first way point (path_home). For this added flags minr and maxr. If the radius are given in sections, and the flags, then the radius is taken from the section. If not specified neither there nor there, then taken the default values of 20 and 40, respectively.

4.6. Additional settings inside schemes, for monsters:

actor_friendly = *true* (ignore actor, until player attack)
npc_friendly = *true* (ignore npc ,until their attack)
friendly = *true* (don't attacks anyone until someone will hit him)
braindead = *true* (ignore any attack)

5. Logic configuration, switching between schemes

In customdata of any character (except for freelancers) should be a section of [logic]. The section must contain one of these keys:

active = active scheme, (always starting first).

cfg = name of _ltx_file_with_settings

If given a cfg, then as setting of character will be used the contents of the file.

Optional parameters:

relation = **neutral** - relation (friend,enemy,neutral)
sympathy = **0** - sympathy multiplier for the player
level_spot = -type of mark on the map: **quest_npc**, **mechanic** , **trader**
trade = *misc\trade_generic.ltx* - sets file with trading parameters

Example. primitive walker settings:

```
[logic]
active = walker
```

```
[walker]
path_walk = walk1
path_look = look1
```

Switching schemes is performed using additional conditions of logic scheme, which are prescribed in the current scheme.

```
[walker]
path_walk        =        walk1
on_info = {+info} camper
```

```
[camper]
```

```

path_walk = walk2
path_look = look2

```

If logic switch between multiple schemes of the same name (such as multiple walker), it is necessary to give a more informative title divided by @ (walker @ day, walker @ alarm, etc.)

Schemes switching conditions:

```

on_info = {=func +info} walker      – by value of function func and a flag info
info on_timer = 1000 | walker        – after 1s
on_game_timer = 10 | walker          – after 1s
on_actor_dist_le = 10 | walker        – when player is closer than 10 meters
on_actor_dist_le_nvis = 10 | walker   – same , but without visibility checking
on_actor_dist_ge = 10 | walker        – players is far more 10 meters
on_actor_dist_ge_nvis = 10 | walker   – same , but without visibility checking
on_signal = signal | walker          – when receiving a signal (the signal can be obtained from the point of
the path, check path settings, or one of the standard signals)
    path_end – NPC has reached the last waypoint
    sound_end – by the end of sound playing (check. sound configuration)
on_actor_in_zone = restrictor_name | scheme – checking if the actor inside of zone (set
    restrictor name)
on_actor_not_in_zone = restrictor_name | scheme – checking if the actor outside of zone (set
    restrictor name)
on_npc_in_zone = story_id | restrictor_name | scheme – if NPC, with story_id inside of zone?
    (set story_id NPC, and restrictor name)
on_npc_not_in_zone = npc_story_id | restrictor_name | scheme – if NPC, with story_id outside of
    zone ?(set story_id NPC, and restrictor name)

on_actor_inside = scheme – zone checks if the player inside (better use function on_actor_in_zone)

on_actor_outside = scheme – zone checks if the player outside (better use function
on_actor_in_zone)

on_offline = – moving stalkers offline condition (supporting condlist)

```

NB: with any of the above options you can work as follows:

```

on_info = {...} %...%
on_info2 = {...} %...%
on_info3 = {...} %...%
and so on

```

This will work too

```

on_info9 = {...} %...%
on_info7 = {...} %...%
on_info5 = {...} %...%

```

Only one rule - All string must be different

```

on_info2=...
on_info2=...

```

This will not work.

NB: all parameters supports condlist:
on_timer = 1000 | {=func +info} walker

5.1. Settings inside of sections

5.1. Configuration of conditions and running functions

Infoportion - conditional flag to mark the event. Has two states - "granted " and "not granted".

Condlist – field to check the status of infoportions and function return values:

{+info =func} – checking conditions

%+info =func% – issuance of infoportion, startig function

on_info = %+info1% – infoportion granted *info1*

on_info = %-info1% – infoportion removed *info1*

on_info = {+info1} – checking if infoportion granted *info1*

on_info = {-info1} – checking if infoportion not granted *info1*

You can check condlist with functions:

on_info = {=func} – checks, that function *func* returned **true**

on_info = {!func} – checks, that *func* returned **false**

on_info = %=func% – running function

5.2. Working with sounds

To add a new sound to prescribe it in the file *script_sound.ltx* or in one of the included files.
In the section *[list]* to register the name of the sound, and create a section of the same name

```
[list]
lvl_sound_name_1
lvl_sound_name_2
...
[lvl_sound_name_1]
type      =      actor
npc_prefix = false
path = characters_voice\scenario\scenario\level\lvl_sound_name_1
shuffle = rnd
idle = 3,5,100
```

type= – type of sound, can be **actor** , **npc** –NPC voice, **3d** – object sound;

path= – path to sound file (NOTE: if the type of sound **npc**, then the path to specify the folder *characters_voice*)

If you do not specify the full file name (*lvl_sound_name_*), then will play all audio files that begin with the specified name;

shuffle= rnd - repeat sound, *seq* - single sound, *loop* - cycled sound

idle = – The first two digits - the delay (in sec.) before re-playing sound, the third probability to play

To play a sound in logic of NPC or another object, you must use the function:
=play_sound(sound_name)

To stop:
=stop_sound(sound_name)

Signals are working in these function
sound_end – end of sound
theme_end – end of sound theme,

If you need to play sound all the time, use function:
=play_sound_looped(sound_name)

Stops cycled sound:
=stop_sound_looped(имя_звука)

Example:

```
[walker@talk]
path_walk = walk1
path_look = look1
on_info = {+info1} %=play_sound(lvl_sound_name_1)%
on_signal = sound_end | nil
```

5.4. Counters

It is possible to use the counters for all events in the game (spawn of the objects, the murder NPC, etc.) and, accordingly, read their conditions. Functions used to create and set the values of the counter:

=set_counter(counter_name:value)	– creates a counter with this value;
=inc_counter(name:value)	– increase value;
=dec_counter(name:value)	– decrease values;

Functions for values checking:

=counter_greater(name:value)	– returns true, if value of counter is bigger than this value
=counter_equal(name:value)	– returns true, if value of counter is equal to this value

Example:

```
[walker@run]
path_walk = walk1
on_signal = path_end | walker@wait %=inc_counter(lvl_scene_fighters)%

[walker@wait]
```

```

path_walk = walk2
path_look = look2
on_info = {=counter_greater(lvl_scene_fighters:5)} walker@fight

[walker@fight]
path_walk = walk3

```

5.3. Cutscenes

You can use restrictor scheme, [sr_cutscene] to control camera starting in cutscene, or this function:

=run_cam_effector(file_path\cam_effector_name:number:true>false) – runs camera effect from player coordinates

file_path – path to camera effect file (\gamedata\anim\camera_effects)

cam_effector_name – name of camera effect file

number – number of camera effect (optional)

true>false – cycled or not?

=run_cam_effector_global(cam_effector_name:number) – runs cam effect from global coordinated (which stored in cam effect file)

=stop_cam_effector(number) – Stops cam effect

5.5. Postprocessings

Use these functions for working with postprocessings:

=run_postprocess(name:number:true>false) – runs postprocess

number – number of postprocess(optional)

true>false – cycled or not

=stop_postprocess(number) – stops postprocess with this number

Example:

```

[logic]
active = sr_idle

```

```

[sr_idle]
on_info = %=run_postprocess(agr_u_fade)%

```

agr_u_fade - name of postprocess

Completed postprocesses:

alcohol	– drunk effect
agr_u_fade	– fade the screen
agr_u_fade_water	– darkening the screen with haze
mar_fade	– fade the screen 2
mar_fade	– gradual darkening and the lightening the screen
blink	– white flash (blink)

5.6. Digital identeficators, story id

Any spawn object can be assigned identeficator story_id. To do this you need to write in the file spawn_sections_ the following logic:

story_id = "name_id" (where name_id – unic name)

Example:

At file spawn_sections_
[zat_vasya]:stalker
\$spawn = "respawn\ zat_vasya"
character_profile = zat_vasya
story_id = zat_vasya

Or at the section logic of spwan element:

[story_object]
story_id = zat_vasya

ATTENTION! There should be no two IDs with the same name.

5.7. Logic Examples

Example: For the character went on the path walk1, but when approaching a player at a distance of 5 meters, switched to path walk2 (but only on condition that he sees the player), you need to write the following:

[logic]
active = walker1

[walker1]
path_walk = walk1
path_look = look1
on_actor_dist_le = 5 | walker2

[walker2]
path_walk = walk2
path_look = look2

Above, we consider the absolute switching of sections.

Before the name of the section in the braces {} you can specify additional conditions, and after the name of the section - the so-called "effects" that are enclosed in percent signs:%%. Effects will only be applied in the case of the section activation. You can not specify a section name, and only specify the conditions and / or effects. Then the old section will remain active, but the conditions and effects are still processed. If all conditions in the curly brackets are not met, the section will not be activated.

Example:

on_actor_dist_le = 5 | {condition} walker2 %effects%

Conditions may include:

<i>+infoportion</i>	- requires the presence of actor infoportion
<i>-infoportion</i>	- requires the absence of actor infoportion
<i>=func</i>	- requires, that <i>func</i> returned true
<i>!func</i>	- requires, that <i>func</i> returned false

Effects may include:

+infoportion - in the case of section activation infoportion will be granted to actor

-infoportion - in the case of section activation infoportion will be removed from actor

=func - in the case of section activation function **func** will start

Multiple condition or effects separates by space bar:

```
on_actor_dist_le = 5 | {+info1 -info2 +info3} walker2 %+info4 =func%
```

You can specify multiple sections, separated by commas. work order in this case - from left to right. After response of the first condition, bypassing stops. In the example below, if you set info1, the scheme will be included walker2, otherwise, if set info2, the scheme will be included walker3, otherwise it will be included walker4:

```
on_actor_dist_le = 5 | {+info1} walker2, {+info2} walker3, walker4
```

In the above field active section of logic, you can also set conditions, such as:

```
[logic]  
active = {=actor_friend} walker@friendly, walker@enemy
```

In logical terms now accepted the keyword never, which means that the condition is false. For example:

```
combat_ignore_cond = {=actor_enemy =actor_has_suit} always, {=actor_enemy} never  
%...effects...%
```

The above structure includes ignoring the combat, if the NPC's enemy - the player in a suit, but disable it if the enemy is a player, without the suit, and the effects (%%) **section of never** will work. So, the selection **section of never** equivalent to the absence of section (failure condition), but the effects of the percent sign will start.

Example with nil section. Section nil removes from the scripted schemes character, monster or object and releases it under the control of the engine. Its needs if a condition is met 1 time and no longer needs to be checked, its saves resources of machines, cause each update check this condition.

```
[logic]  
active = sr_idle  
  
[sr_idle]  
on_actor_inside = nil %+esc_actor_inside%
```

At the entrance of the actor inside the restrictor infoportion will be granted, and restrictor will go to the section nil, no more checking for the player presence.

NB: Its impossible to return the object from nil section to scripts control, so be caution

Example of some sort difficult logic:

```
[logic]
active = walker
on_hit = hit
on_death = death

[hit]
on_info = %+alert%

[death]
on_info = %+alert +trup3%

[walker]
path_walk = walk_svoboda3
path_look = look_svoboda3
combat_ignore_cond = {-alert}
on_timer = 25000 | remark

[remark]
anim = idle
snd = stalker_talk_kampfire
no_move = true
no_rotate = true
on_hit = hit
on_death = death
combat_ignore_cond = {-alert}
```

Let's see by steps. Initially, stalker works by walker scheme. He ignores the fight until granted infoportion alert. He waits 25 seconds, then moves into the scheme remark. In the remark he plays idle animtion, speaks on some optional themes, does not rotates and moves, just ignores the fight. If he will take a hit (on_hit) or becomes dead (on_death), infoportion alert will be granted and he will no longer ignore the battle (of course, if he is a corpse, it's not will help him, but there are 3 stalkers in the scense, and they will start to fighting all). If you kill him, it will also set infoportion trup3 who reported that the stalker is killed.

Here is an enemy logic:

```
[logic]
active = walker

[walker]
path_walk = soldier_walk1
path_look = soldier_look1
combat_ignore_cond = true
team = assault_group
on_signal = assault | camper

[camper]
```

```

path_walk = soldier_walk1_2
path_look = soldier_look1_2
radius = 5
on_info = {+trup1 +trup2 +trup3} walker2

```

```

[walker2]
path_walk = soldier_walk1_3
path_look = soldier_look1_3

```

He goes by the walker scheme, ignoring the combat (at any situation). He is part of a group assault_group. When he comes to the final point (where he is synchronized with the rest of the group (its described at the paths)) and receives an assault signal, then goes into the camper scheme. In this scheme, it has not restrictions combat_ignore, so he starts shooting at the enemy. Once all three enemies are killed, every one of them, dying sets infoporshn trup1, trup2 or trup3 and when all three will be killed, then he will switch to a scheme walker2 (returns to the fireplace).

6. Logic schemes space_restrictor

To exclude the situation where the actor slips through the restrictor and he did not have time to work, try to place the restrictor so, that the minimum width will be greater than 2 meters.

6.1.Scheme sr_idle

Purpose of the scheme - to activate another scheme while triggering one of the standard terms of logic. By itself, the scheme does nothing. Example of settings restrictor:

```

[logic]
active = sr_idle

[sr_idle]
on_actor_inside = nil %+esc_actor_inside%

```

Note, that after launching active schemes switches to nil, for not continuing useless checks for each update. You cannot set a nil. Often, this scheme works with the spawner, the restrictor gives infoportion, at the entrance to the zone, and spawner on it already spawns someone.

6.2. Section sr_no_weapon

Player holster mode inside specific zones.

Restrictor example:

```

[logic]
active = sr_no_weapon

[sr_no_weapon]

```

6.3. Section sr_light

Zone where NPC's head lamps always turned on

```
[logic]  
active = sr_light
```

```
[sr_light]  
light_on = true/false (on/off)
```

Also works together with condlist:

```
[logic]  
active = sr_light
```

```
[sr_light]  
light_on = true/false (on/off)  
on_info = {+info1} section %+info2%
```

6.4.Scheme sr_particle

This system plays back particles (both static, and moving in a specified location)at specified time

1) For particle system with camera path:

```
[sr_particle]  
name = explosions\campfire_03      -name of particle system  
path = particle_test.anm           -name of camera path  
mode = 1                           (necessarily!!!)  
looped = true/false                -particle cycle flag
```

2) With standard patrol path:

```
[sr_particle]  
name = explosions\campfire_03      -name of particle system  
path = part_points                 -name of patrol path  
mode = 2                           (necessarily!!!)  
looped = true/false                -particle cycle flag
```

In the waypoints you can set the flag s = name of sound theme and d = delay before playback (given in milliseconds. If not specified, 0). s - the name of the sound theme in sound_themes.ph_snd_themes which will be randomly selected to play the sound while playing particle. Sound plays only once...

Result = particles plays at all waypoints simultaneously (or with a delay, see above).

When looped = true after playing particles, they will be run from beginning, but without delay. Particle_end signal will not be granted. When looped = false signal will be granted when all particle sources played.

Condlist Supported.

If the restrictor goes to another section, it will automatically cease playing of particles and silent sounds with them. This restrictor is an object that tracks particles and there is no need for a player to go inside.

6.5.Scheme sr_timer

Example:

[logic]

active = sr_timer@1

[sr_timer@1]

type=dec

start_value = 10000

on_value = 0 | sr_timer@2

[sr_timer@2]

type = inc

on_value = 15000 | nil %+info1%

Description:

type - type of counter (inc\dec).

start_value - in real milliseconds

The transitions from section sr_timer can be both on the usual conditions (on_timer, on_info) and on the specific condition on_value. In general on_value can be used to produce any action, depending on the state of the counter.

For example:

on_value = 5000| %+info1% | 1000| %+info2%

6.6. Scheme sr_psy_antenna

Psy zones effects(Radar/Yantar)

eff_intensity = - inc/dec at % from basis radiation value.

hit_intensity = - inc/dec at % from basis hit

Example, zone that, increase 70% of psy:

[logic]

active = sr_psy_antenna

[sr_psy_antenna]

eff_intensity = 70

hit_intensity = 70

Example, zone that, decrease 30% of psy:

[logic]

active = sr_psy_antenna

[sr_psy_antenna]
intensity = -30

6.7. Scheme sr_cutscene

In our engine we can use camera (Cutscene)

Example:

[logic]
active = sr_idle

[sr_idle]
on_info = {!black_screen + agru_nvidia_presentation} sr_cutscene@cam1

Here we checking infoportion **agru_nvidia_presentation** and function **black_screen** and jump to the section **sr_cutscene@cam1**

[sr_cutscene@cam1]
point = agru_nv_camera_walk – point **walk**, where the player will be placed after the camera

look = agru_nv_camera_look – point **look** where be aimed actor camera after the main camera

cam_effector = scenario_cam\Agraprom_underground\camera1_0_904 –path to camera file

on_signal = cameff_end | sr_cutscene@cam2 – end signal of the camera and transition to the next section

global_cameffect = true/false – camera playing flag (from global coordinates, not player's, by default - false)

[sr_cutscene@cam2]

Attention! Camera from the restrictor will work only, when:

- 1) at the time of its launch the player is inside of restrictor
- 2) the restrictor has type NONE default restrictor in Level editor

7. Additional logic setting for other objects.

For all physical objects there is a section ph_idle, supporting condlist in which you can, if necessary transit objects

7.1. Door working scheme, section ph_door

NB! For double gates all is similar.

locked = *false\true* Door locked? default – false.

closed = *false\true* Door closed? default - true

tip_open = (if *locked* == false, then *tip_door_open*, otherwise *tip_door_locked*)
Tip when you aim closed door.

tip_close = (if *locked* == false, then *tip_door_close*, otherwise empty place)
Tip when you aim opened door.

snd_init = Scheme activation sound.

snd_open_start = Door open sound.

snd_close_start = Door close sound.

snd_close_stop = After door is closed sound.

Examples:

If you want to make a door that, when some event happens opens with a click sound, you can use a field *snd_init* and switching schemes. In the example below when you turn on the scheme *ph_door @ unlocked* plays *snd_init*, means *trader_door_unlock*:

```
[logic]
active = ph_door@locked

[ph_door@locked]
locked = true
snd_open_start = trader_door_locked
on_info = {+esc_trader_can_leave} ph_door@unlocked

[ph_door@unlocked]
locked = false
snd_init      = trader_door_unlock
snd_open_start = trader_door_open_start
snd_close_start = trader_door_close_start
snd_close_stop = trader_door_close_stop
```

7.2. Button working scheme, section *ph_button*

Pressing the button toggles the sections and sets infoportion

```
[logic]
active = ph_button@locked

[ph_button@locked]
anim_blend = false
anim = button_false
on_press = ph_button@unlocked %+cit_jail_door_opened%
```

on_press – What's happening with pressing of button

anim – button pressing animation
anim_blend – smooth, polished animation.(true/false)

***tooltip** – intended to set the help text when you aim over the button. Textual clues needed to at least be clear that this device can be pressed.

Button config example:

```
[logic]
active = ph_button@active

[ph_button@active]
anim = lab_switcher_idle
tooltip = tips_labx16switcher_press
on_press = ph_button@deactivated %+terrain_test%

[ph_button@deactivated]
anim = lab_switcher_off
```

In order that message has not lost relevance in various settings of keyboard, message should be written using the tokens. For example:

```
<string id="tips_labx16switcher_press">
  <text>to shut down magic device press ($$ACTION_USE$$)</text>
</string>
```

Here is an example of button that works only with special condition:

```
[logic]
active = ph_button@locked

[ph_button@locked]
anim = button_false – button denied animation
on_info = {+val_prisoner_door_unlocked} ph_button@unlocked
on_press = ph_button@unlocked %+val_prisoner_door_unlocked%

[ph_button@unlocked]
anim = button_true
on_info = {-val_prisoner_door_unlocked} ph_button@locked
on_press = ph_button@locked %-val_prisoner_door_unlocked%
```

7.3. Gate working sheme, section ph_gate:

Same as ph_door, but for double gates! (eg: door_darkvaley_outdoor_01)

Instead parameters closed and locked now used parameters:

state: door initialization state (by default none)

-open

-closed

-none - default

locking: door blocking (by default none)

-stick - (in development process)

-soft - door blocked by physics force ,you can open it by yourself or with the car

Available states:

open - block in open state
closed - in closed state
none
hard - blocked by borders ,you can only break the gate

Available states:

open
closed
none

none - door is not blocked

General parameters:

left_limit, right_limit - [0-180] (default - 100)
breakable - (true/false)

Sound parameters: similar to ph_door

Example:

```
[ph_gate@locked] ; Blocked in opened state (unbreakable)
state = opened
locking = soft
left_limit = 130
right_limit = 60
breakable = false
```

```
[ph_gate@opened]
state = opened
locking = stick
```

```
[ph_gate@closed]
state = closed
```

7.4. Code locks, ph_code:

Entering of code gives infoportion

```
[logic]
active = ph_code@lock
```

```
[ph_code@lock]
code = 1243
on_code = %+infoportion%
```

7.5. Push physics object, ph_force

The scheme allows you to kick the object in the specified direction. Prescribed in the custom date of object

force = Force (in dead Coons :))

time = while a force to the object (in seconds)

***delay** = delay (in seconds) before using force

point = the name of the patrol path, a point which will be used as a target (where to send the object)

point_index = index of patrol path point toward that the object flies.

7.6. Object throwing denying, **ph_heavy**

Prescribed in the physical objects that have been banned for throwing by Burer and Poltergeist. For example, if they must lie in a specific place (story document type) or too cumbersome in size, so they can't be nice to throw.

Write in CustomData:

[ph_heavy]

7.7. Hanging physics, **ph_oscillate**

Smooth hanging physics (lamps, hanging zombies etc.)

Example:

[ph_oscillate]

joint = **provod** – the name of the bone to which the force will be applied (check mesh in actoreditor for bones)

force = 5 – the force (in Newtons)

period = 1000 – force allaying time.

Force is applied to the bone of object with a linear increasing. During the task of time period the force will grow from 0 to the declared value. After that comes a pause (the force is not applied) during the time period / 2. After the end of the pause force is applied in the same way as in the beginning, but in the opposite direction.

7.8. Reaction on sound

You can do that NPC or monster will be react to a certain volume of sound.

on_sound = **story_id** | **sound_type** | **distance** | **sound_power** | {conditions} **section** %effects%

Types of Sound:

sound_type:

WPN_hit - bullet hit sound

WPN_reload - wpn reloading sound

WPN_empty - shot with empty clip (no bullets)

WPN_shoot - Shot sound

MST_die - dying sound

MST_damage - taking damage sound

MST_step - Steps sound

The transition will take place if the object will hear the sound (**sound_type**) from object(**story_id**) with the distance <=

distance and sound power >= **sound_power**.

this scripting supported:

on_sound1 =

on_sound2 =

if you need to catch the sound with any power, then **sound_power** set to 0.

If you need to catch the sound on any distance , then **distance** set to 10000.

Example:

[logic]

active = mob_walker@spawn

[mob_walker@spawn]

path_walk = zat_b38_sleeper_bloodsucker_1_walk_1

path_look = zat_b38_sleeper_bloodsucker_1_look_1

on_info = mob_walker@sleep

[mob_walker@sleep]

on_sound = zat_cop_id|WPN_shoot|10|0.9| mob_home@fight ; Hears everything, except a pistol with a silencer

on_sound2 = zat_cop_id|WPN_hit|10|0| mob_home@fight

on_sound3 = zat_cop_id|MST_damage|10|0.9| mob_home@fight

on_sound4 = zat_cop_id|MST_step|10|0.5| mob_home@fight

on_sound5 = zat_cop_id|MST_die|10|0| mob_home@fight

on_sound6 = zat_cop_id|WPN_empty|10|0.3| mob_home@fight ;Hears almost within 3 meters

on_sound7 = zat_cop_id|WPN_reload|10|0.3| mob_home@fight;Hears within 3 meters

8. Controlling some features

8.1. NPC reaction settings, meet_manager

Syntaxes:

[logic]

active = walker

[walker]

meet = meet

[meet]

meet_state	= 30 state@sound 20 state@sound 10 state@sound
meet_state_wpn	= 30 state@sound 20 state@sound 10 state@sound
victim	= 30 nil 20 actor
victim_wpn	= 30 nil 20 actor
sound_start	= nil ;Starting sound when sees the player
sound_start_wpn	= nil ;Starting sound when sees the player with weapon
sound_stop	= nil ;Sound when players stands still
use	= self
use_wpn	= false
zone	= name state@sound
meet_dialog	= dialog_id
synpairs	= state@sound state@sound
precond	= visibility/usability

<i>abuse</i>	= true/false
<i>trade_enable</i>	= true/false ;default true. Can we trade with npc?
<i>allow_break</i>	= true/false ;default false.Can we quit from dialog?
<i>quest_npc</i>	= true/false ;default false. Is it quest npc?

The whole setup of the meetings will now be made in a separate section. In the section of logic or in the current scheme, you can specify what kind of section to the setting you want to use. Section, which is specified in section logic will affect the handling of meetings with free wandering stalkers.

A list of parameters:

meet_state, meet_state_wpn - defines animation and voice acting of character, depending on the distance from the actor. For the case if the actor is armed or unarmed, respectively.

victim, victim_wpn - specifies the object to which will have to watch the character. Possible options: nil - will not look, actor - look at the player,

story_id - Number of Story ID character in which you want to watch.

use, use_wpn - setting usability of the character. There are three options: true, false, self. When self NPC itself will use a player as soon as you reach

zone - Provides a set of names with a restrictor, as well as animations and voice acting that NPCs will act out if the player will be seen in the restrictor

meet_dialog - starting NPC dialogue.

synpairs - Contains a set of pairs body condition @ sound theme.
If for some set of conditions that the meeting will act this states, then this sound theme and condition will be randomly synchronized animated states of body.

abuse - default true, if false, then not usable opponent will not be offended.
Any line (in the general scheme they are written in small letters) can be defined by condlist.({+ Info1-info2} ward% + info%)

quest_npc - default

To ease the meeting configuration, it's possible to use set of simple defaults:

```
[walker]
meet = default_meet
```

No need to write [default_meet] itself. All settings will take out from default automatically.

Now about how to use this constructor to make any response to the actor (In all the examples in green color states of state_manager, blue - sound themes):

Situation 1

Character in the distance call us with a hand, when you approach requests to secure the weapon, then agrees to talk.

```
[meet]
meet_state      = 50| hello@talk_hello| 20| wait@wait| 10| ward@wait
meet_state_wpn  = 50| hello@talk_hello| 20| threat@threat_weap
victim          = 50| actor
victim_wpn      = 50| actor
use             = true
use_wpn         = false
```

Situation 2

Stalker seeing a player asks to secure the weapon. After that comes and speaks to us. If we start to go away or get weapon - starts to shoot at us.

```

[meet]
meet_state           = 50| {+info} threat_fire %=killactor%, walk@ {+info} talk_abuse, wait | 10 |
walk %+info%; wait | 2 | threat;state
meet_state_wpn       = 50| {+info} threat_fire %=killactor%, threat@ {+info} talk_abuse, wait
victim               = 50| actor
victim_wpn           = 50| actor
use                  = {-info2} self, false
use_wpn              = false

```

Here: info – Infoportion, which indicates that we have secured already our weapon and were close enough to the NPC

Info2 – is set in dialogs, and it tells that character told us everything he wanted

Killactor –xr_effects function

Situation 3

The character goes on patrol path on the outpost camp. If the player has access to the camp - his greets player ,and give the way in, or at first discouraged, and if the player made his way to the camp - attacks him, depending on whether the player has access to the camp or not.

```

[camper]
path_walk = path_walk
path_look = path_look
meet = meet

[meet]
meet_state           = 30| {+info} wait, threat@ {+info} talk_hello, threat_back
meet_state_wpn       = 30| {+info} wait, threat@ {+info} talk_hello, threat_back
victim               = 30| actor
victim_wpn           = 30| actor
use                  = true
use_wpn              = true
zone                 = warnzone| {-info} threat@ {-info} threat_back|kampzone| {-info} true@ {-
info} talk_abuse
meet_dialog           = {+info} dialog1, dialog2

```

Here:

True – instead of animation, attack the player.

Info – Infoportion, which indicate that player has access to the camp

Warnzone – restrictor, (NPC will warn us)

Kampzone – restrictor, (NPC will shoot us)

Dialog1 – Starting dialog (player has access to the camp)

Dialog2 – Starting dialog (player has not access to the camp)

Default parameters:

```

meet_state           = 30|hello@hail|20|wait@wait
meet_state_wpn       = 30|backoff@threat_weap
victim               = 30|actor
victim_wpn           = 30|actor
use                  = true
use_wpn              = false

```

syndata = *hello@hail|backoff@threat_weap*

NB: If you want that stalker don't speaks to the player in this section, you must write:

meet = no_meet

8.2. Smartcovers (CS-COP)

Smartcovers - its spawn object that controls the animations of NPC.

Used to create scenes with complicate animations. Type of smartcover (animations which can be by NPC in smartcovers) depends on the parameter **description** (selected in Level editor).

Combat smartcovers, manage combat animations, they can be used by NPC in the universal battle schemes. You may use combat smartcovers to build scripted scenes.

Animation set used by NPC in smartcovers, depends on the **loopholes**.

NPC can attack from smartcover only if the enemy is within range of **loophole**.

Names **loopholes** for combat smartcovers look in the file: **Battle loopholes.xls**

Animation of smartcovers are configured similarly as combat ones, but special animations is used only for scripted scenes. Names of loopholes for animated smartcovers look in the file: **Lead loopholes.xls**

[smartcover]

cover_name = cover_1

– name of smartover (necessary!)

loophole_name = lh1

– name of loophole (depends on smartcover type)

cover_state = fire_target

– NPC state in smartcover:

fire_target – shoots the target,

fire_no_lookout_target – shoots the target being covered,

idle_target – hide,

lookout_target – looks from the
hideout,

default_behaviour – hide, lookout, if there a target, attacks

use_in_combat = true

– использование смарткавера в бою, если не поставить этот параметр NPC будет срываться в универсальную боевую схему

target_enemy =

= <path_name>

– target where to shoot: **actor** or **story_id** target_path

idle_min_time

– shoot the specified way point

idle_max_time

– timings between anims

lookout_min_time

– timings between anims

lookout_max_time

– timings between anims

ATTENTION! Before you set a loophole in smartcover, make sure that loophole with such name is in this smartcover (check the files **Battle loopholes.xls**, **Lead loopholes.xls**).

Standart smartcovers signals:

enemy_in_fov – enemy inside the loophole

enemy_not_in_fov – enemy outside the loophole

8.3. Text outputting

In order to mark something urgent and important. For example, an urgent message that you must leave the area, you can use text. He will be above the crosshair.

An example of adding text:

```
[logic]
active = sr_idle
[sr_idle]
on_info = %=add_cs_text(agru_cs_warning)%
; Where agru_cs_warning = id strings
```

Пример удаления текста:

```
[logic]
active = sr_idle
[sr_idle]
on_info = %=del_cs_text()%
```

8.4. Activation/Deactivation anomalies

Some specific abnormalities (eg agru_firtude) can be switched on and off by functions **=disable_anomaly(game_story_id аномалии)** и **=enable_anomaly(game_story_id аномалии)**.

ATTENTION! Anomaly activated by default

Turning off:

```
[logic]
active = sr_idle

[sr_idle]
on_info = %=disable_anomaly(story_id)%
```

Turning on:

```
[logic]
active = sr_idle

[sr_idle]
on_info = %=enable_anomaly(story_id)%
```

8.5. Working with text

All the texts creates in XML files, which placed in
\\gamedata\\configs\\text\\localization_folder

Inside file XML

```
<string id="lvl_item_name">
    <text>Some item blabla</text>
</string>
```

The ID of the text should be called by the rules:

The first write abbreviation level («agr» - agroprom), in cases where the text does not refer to a certain level, you should add st_

Then write the abbreviated name of the scene, character or object. The last write what this text is:

name – title

text – task text

descr – description

8.6. Disabling post combat

Write this key at logic section **[logic]** - **post_combat_time** with value «0, 0»

[logic]

active = walker

post_combat_time = 0, 0

8.6. Physics objects settings

[collide] - object collision settings;

ignore_static - disable collision with static;

small_object - the object is *small_object*, (you need it to configure conflict with physics)

ignore_small_objects

9. Syntaxes of LTX files

Similar to **INI** files

9.1. Sections,keys

[section]

key = value1

9.2. Inheritance

Section in the LTX files can inherit from other sections, for this after the closing parenthesis is necessary to put a colon and a list, separated by a comma, the names of sections that you want to inherit. When you inherit all the keys with the values go from section to section of the inherited file. Section heir must be inherited from lower sections.

Example:

[section_1]

key1 = 1

key2 = 2

key3 = 3

[section_2]: section_1

[section_3]: section_1, section_2

9.3. Includes

You can put LTX file inside another LTX file.

#include "file_path\included_file.ltx"

Type the folder which contains the file in what you need to place another file.

author: GSC Gameworld
description: game designers manual
english translation: Viking aka Darkman