

Оглавление

1. Система расстановки путей.....	3
1.1. Более подробное описание путей.....	4
2. Схемы поведения сталкеров.....	7
2.1. Схема walker.....	7
2.2. Схема camper.....	7
2.2.1. Схема sniper.....	8
2.3. Схема remark.....	9
2.4. Схема patrol.....	9
2.5. Схема sleeper.....	10
2.6. Схема kamp.....	10
2.7. Схема wounded (раненный).....	10
2.8. Дополнительные настройки внутри схем.....	12
3. Настройки логики по событию.....	12
3.1. Секция combat.....	12
3.2. Секция death.....	12
3.3. Секция hit.....	12
3.4. Секция danger.....	13
3.5. Дополнительные секции.....	13
3.5.1. Секция dont_spawn_character_supplies.....	13
3.5.2. Секция dont_spawn_loot.....	14
3.5.3. Секция spawner.....	14
3.5.4. Секция known_info.....	14
4. Схемы логики для монстров.....	14
4.1. Схема mob_walker.....	14
4.2. Схема mob_remark.....	15
4.3. Схема mob_combat, mob_death.....	15
4.4. Схема mob_jump (монстр-пружинка).....	15
4.5. Схема mob_home.....	16
4.6. Дополнительные настройки внутри схем, для монстров.....	16
5. Настройка логики и переключение между схемами.....	16
5.1. Настройки внутри секции.....	18
5.1. Настройка условий и запуск функций.....	18
5.2. Работа со звуками.....	19
5.4. Счетчики.....	19
5.3. Катсцены (ролики на движке).....	20
5.5. Постпроцессы.....	20
5.6. Числовые идентификаторы, story id.....	21
5.7. Примеры логики.....	21
6. Схемы логики space_restrictor.....	24
6.1. Схема sr_idle.....	24
6.2. Секция sr_no_weapon.....	25
6.3. Секция sr_light.....	25
6.4. Схема sr_particle.....	25
6.5. Схема sr_timer.....	26
6.6. Схема sr_psy_antenna.....	26
6.7. Схема sr_cutscene.....	27
7. Набор дополнительных настроек логики у разных объектов.....	27
7.1. Схема работы двери, секция ph_door.....	28
7.2. Схема работы кнопки, секция ph_button.....	28
7.3. Схема работы ворот, секция ph_gate.....	29

7.4. Кодовые замки, <code>ph_code</code>	30
7.5. Толкнуть физический объект, <code>ph_force</code>	30
7.6. Запретить швырять объект, <code>ph_heavy</code>	31
7.7. Раскачивание физики, <code>ph_oscillate</code>	31
7.8. Реакция на звук.....	31
8. Управление некоторыми фичами.....	32
8.1. Настройка реакции NPC, <code>meet_manager</code>	32
8.2. Смарткаверы.....	35
8.3. Вывод текста на экран.....	35
8.4. Включение и выключение аномалий.....	36
8.5. Работа с текстами.....	36
8.6. Отключение посткомбата.....	36
8.6. Настройки физических объектов.....	37
9. Синтаксис файлов LTX.....	37
9.1. Секции, ключи.....	37
9.2. Наследования.....	37
9.3. Инклюды.....	37

1. Система расстановки путей

В точках путей можно задавать флаги, изменяющие поведение персонажа. Флаги задаются прямо в имени waypoint-a, например, для точки с именем "wp00":

wp00|flag1|flag2

Флаги точек пути path_walk:

a=state

Выбирает состояние тела при перемещении (Только из раздела –Ходячие состояния)

Список состояний можно взять в gamedata\scripts\state_lib.script

p=percent

Вероятность остановиться в точке в процентах (0 – 100). По умолчанию 100, т.е. сталкер никогда не проходит мимо точек остановки.

sig=name

Установить сигнал с именем name сразу по прибытию в точку (до поворота) для последующей его проверки с помощью поля on_signal логической схемы. Если нужно установить сигнал после поворота – используйте соответствующий флажок пути path_look.

Флаги точек пути path_look:

a =state

Выбирает состояние тела при стоянии (или сидении) на месте. (Из разделов Стоячие и Сидячие состояния)

Список состояний можно взять в gamedata\scripts\state_lib.script

t=msec

- время в миллисекундах, которое персонаж должен смотреть в заданную точку.

‘*’ – бесконечное время. Допустимы значения в диапазоне [1000, 30000], по умолчанию – 5000.

Для конечных (терминальных) вершин пути path_walk, у которых не более 1-й соответствующей точки path_look, значение t всегда считается бесконечным и его явно задавать не нужно.

sig=name

После поворота в точку path_look, установить сигнал с именем name.

syn

Наличие флажка задержит установку сигнала до тех пор, пока в точку с флажком syn не придут все персонажи с данным team-ом (team задается в виде текстовой строки в customdata). До тех пор, пока остальные персонажи не придут, ожидающей персонаж будет отыгрывать свою idle анимацию.

sigtm=signal

Устанавливает сигнал при вызове time_callback-a state manager-ом. Соответственно, если t=0, то сигнал будет установлен после отыгрыша init анимации. Это используется, например, с анимацией press, которая состоит из двух частей: 1 - нажимаем на кнопку, 2 - опускаем руку.

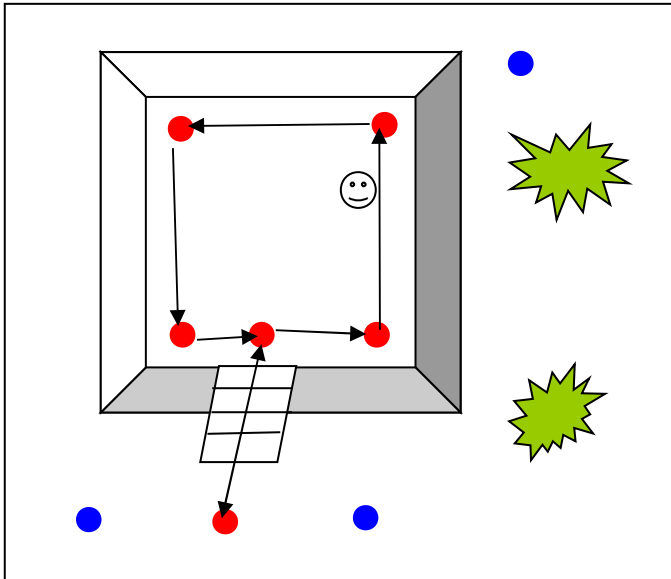
В пути path_look можно сделать: wp00|a=press|t=0|sigtm=presed

А затем переключить схему: on_signal = presed | другая_схема

1.1. Более подробное описание путей.

Walker.

Настройка:



На карту для каждого walker-а нужно поставить:

- 1) Путь `path_walk`, по которому walker ходит.
- 2) Путь `path_look`, состоящий из точек, в которые walker смотрит.

Walker-ов может быть 1 или больше. Они могут действовать независимо, или взаимодействовать друг с другом.

[walker]

`team = ...`

имя команды, произвольная текстовая строка. Все walker-ы в одной команде должны иметь один и тот же `team`. Желательно в `team` задавать имя уровня и имя места, где стоят walker-ы, например: `escape_bridge`, `escape_factory`, это уменьшит шанс ошибиться и дать разным командам общее имя.

`path_walk = ...`

имя пути, описанного в п. 1

`path_look = ...`

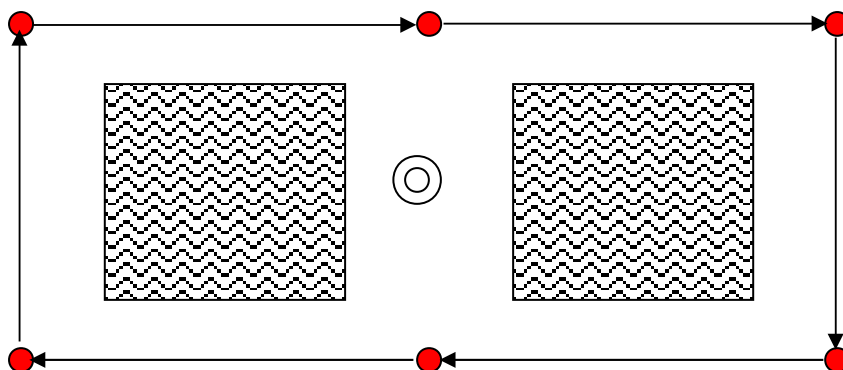
(не обязательно) имя пути, описанного в п. 2. Если персонаж должен только **ходить** по маршруту, `path_look` можно не задавать.

Если персонаж должен стоять на месте, то ему задается одна точка пути `path_walk` и как минимум одна точка пути `path_look`

Правила расстановки флажков в путях рассмотрим на нескольких примерах:

Пример 1:

Персонаж патрулирует территорию вокруг двух домиков. Маршрут строится следующим образом:



Как сделать, чтобы персонаж между определенными точками бежал или крался? Для этого в пути **path_walk** существуют флажки.

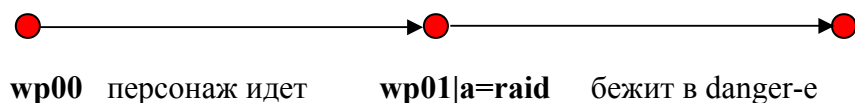
У каждого вейпоинта есть имя: wp00, wp01 и т.д.

Флажки задаются в имени. Их нужно отделять от самого имени с помощью символа '|'. Пишется **a=anim**, где **anim** – название анимации из пункта 2.4.4. настоящей документации. Если мы

напишем **a=threat** то персонаж пойдет в состоянии danger, если **a=raid** то побежит с оружием наизготовку и т.д.

NB: В точках пути **path_walk** используются анимации ТОЛЬКО из раздела «Ходячие состояния»!

Пример 2:



Разговор персонажа.

Чтобы персонаж говорил, перемещаясь по маршруту, нужно определить в каждой точке список тем, на которые он может говорить. Для этого существуют следующие поля:

s = имя_звуковой_схемы (по умолчанию звук отключен). Несколько тем можно перечислять через запятую.

Пример 3:



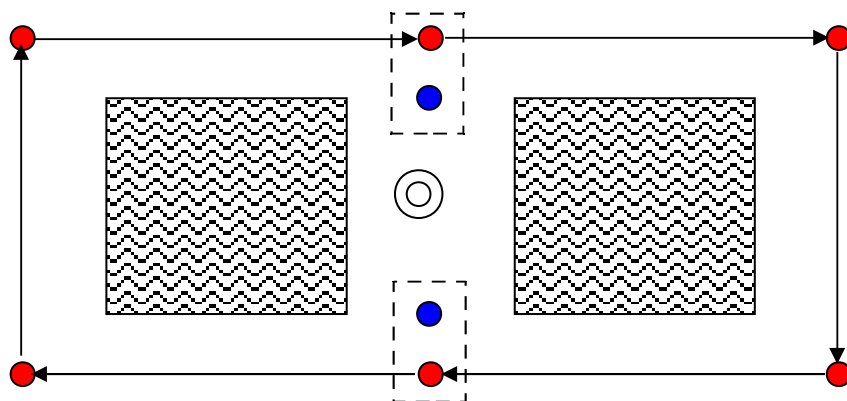
В примере 3 используется только поле **s**, чтобы задать тему разговора, и флажок **sc**, чтобы показать, что звук проигрывается не разово, а периодически.

Остальные параметры (**sp**, **sf**, **st**) задавать НЕ РЕКОМЕНДУЕТСЯ, значения по умолчанию приемлемы для большинства скриптов.

Параметр **sa** также использовать НЕ РЕКОМЕНДУЕТСЯ. Если нужно стартовать звук одновременно с анимацией, лучше воспользоваться полями пути **path_look**, о котором будет написано ниже в этом документе.

Если персонаж не только **ходит** по маршруту, но должен также останавливаться и играть анимации, нужно задать ему путь **path_look**.

Пример 4: усовершенствуем пример 1, чтобы персонаж, проходя мимо проема между домами, останавливался и заглядывал в него:



Что добавилось в этом примере? Путь **path_look** с двумя точками. Связь между точками этого пути рекомендуется сразу же удалить в редакторе, поскольку она все равно не используется.

Далее, в точках путей **path_walk** и **path_look**, которые обведены на рисунке пунктирной линией, в редакторе ставим общие флажки. Например, в верхней паре точек ставим флажок 0, а в нижней паре точек – флажок 1.

Теперь персонаж будет останавливаться в точках **path_walk**, помеченных флажком, и смотреть в точку **path_look**, помеченную тем же самым флажком.

Если точка **path_walk** не помечена флажком, персонаж проходит ее не останавливаясь.

Одной точке **path_walk** может соответствовать несколько точек **path_look**. Тогда персонаж выберет случайно одну из подходящих точек.

По аналогии с **path_walk**, в точках пути **path_look** можно использовать различные флажки, меняющие поведение:

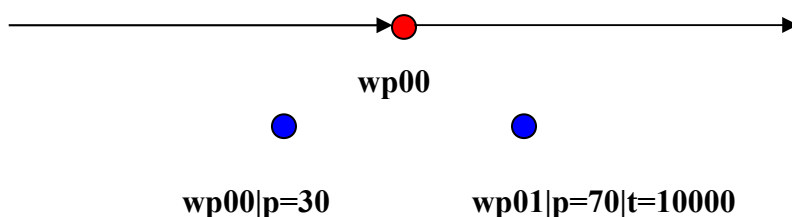
$p = 100$ – вероятность, с которой персонаж посмотрит именно в эту точку. Значения p всех подходящих точек суммируются, т.е. если у одной точки $p = 100$, а у другой 300, то персонаж посмотрит в первую с вероятностью 25%! (т.е. 100 из 400).

Во избежание путаницы, рекомендуется задавать p так, чтобы их сумма составляла 100.

По умолчанию у всех точек $p = 100$.

t = время, на которое персонаж задержится в этой точке (по умолчанию 5000 мсек)

Пример 5:



В этом примере проходя через точку **wp00**, персонаж с вероятностью 30% посмотрит в точку **wp00** в течение 5 секунд, но с вероятностью 70% посмотрит в точку **wp01** в течении 10 секунд.

По умолчанию при остановках персонаж играет анимацию *idle*, если он не в состоянии *crouch*, либо анимацию *hide*, если он в состоянии *crouch*.

Если требуется другая анимация, можно ее указать с помощью флажка:

a = имя_анимации (по умолчанию *idle*).

Пишется *a=anim*, где *anim* – название анимации из пункта 2.4.4. настоящей документации. Если мы напишем *a=hide*, то персонаж сядет в состоянии *данжер*, если *a=guard*, то встанет с оружием наизготовку и т.д.

NB: В точках пути *path_look* используются анимации ТОЛЬКО из раздела «Стоячие и сидячие состояния»!

2. Схемы поведения сталкеров.

Есть определенный набор схем, которые описывают поведение персонажа. Они прописываются у него в *custom_data* или, в случае смарта, в соответствующих файлах, описывающих работы данного смарта. Ниже приведен перечень этих схем.

2.1. Схема *walker*

Это базовая схема, по которой персонаж, перемещается по патрульному пути (*path_walk*) и останавливается в определенных точках и выполняет соответствующие действия.

* - необязательный параметр

[*walker*]

path_walk = <имя пути>

- основной путь, по которому ходит NPC

**path_look* = <имя пути>

- путь, куда смотрит NPC. В точках *path_walk*, которым соответствуют точки пути *path_look* (стоят одинаковые флажки), персонаж останавливается и смотрит в определенную точку. Не обязательный, только если в пути *path_walk* больше одной точки пути

**team* = <имя группы>

- команда для синхронизации

* *def_state_moving1* =

- состояние (анимация), в котором сталкер движется к первой точке пути, если она близко (*patrol* по умолчанию)

* *def_state_moving2* =

- состояние, в котором сталкер движется к первой точке пути, если она не слишком далеко (*rush* по умолчанию)

* *def_state_moving3* =

- состояние, в котором сталкер движется к первой точке пути, если она далеко (*sprint* по умолчанию)

* *def_state_standing* =

дефолтное состояние в котором он стоит и смотрит на точку, если в этой точке не задана другое состояние (*guard* по умолчанию).

В схеме читается стандартный сигнал:

path_end – NPC дошел до конечной точки пути

2.2. Схема *camper*

Свойства кемперов:

- кемпер стоит на точке и смотрит в направлении, куда Вы его поставили в редакторе или передвигается по патрульным путям

- кемперы переключаются на универсальный комбат, только если видят врага ближе чем в 30 метрах. Если он выжил, он возвращается в состояние кемпера.

- В любых других случаях действуют по собственной скриптовой схеме. Если видим врага -стреляем. Если слышим дэнжер - то смотрим в направление в данжере. Если видим гранату - убегаем от гранаты. Если видели врага, а враг исчез, то смотрим в точку, где видели последний раз врага.

- кемперы не сражаются в движении. Если они видят врага - они останавливаются, стреляют, а потом продолжают движение.

[camper]

path_walk = *patrol_path*

**path_look* = *patrol_path*

– не обязательный, только если в пути *path_walk* больше одной точки пути

**radius* =

– расстояние в метрах, если расстояние между кэмпером и противником меньше указанного, кэмпер уходит в универсальный комбат. По умолчанию этот радиус равен 20 метрам.

**no_retreat* = *true*

– персонаж при виде врага не будет ломиться на ближайшую точку *path_walk*, а сразу перейдет в режим убивания. Нужно это в том случае, если вы хотите сделать сценку, когда одни ребята наезжают на других. Ставьте кемперов с вышеуказанным флажком. Они идут по своим патрульным путям и выносят врагов.

**def_state_moving* =

– анимация, в которой сталкер движется в ближайшую точку пути при враге (по умолчанию *sneak*)

**def_state_moving_fire* =

– состояние (анимация), в сталкер отстреливается от врага, во время движения на ближайшую точку пути (*sneak_fire*)

**def_state_campering* =

– состояние, в котором сталкер ожидает врага, находясь на пути (*hide*)

**def_state_campering_fire* =

– состояние, в котором сталкер отстреливается от врага, находясь на пути (*hide_fire*)

**attack_sound* =

имя_звуковой_темы

Возможность переопределять снайперам/кемперам звук атаки. По дефолту он равен звуковой теме "fight_attack". Можно изменить на любое другое (для сценических потребностей) либо вообще отключить, прописав в секции кемпера: *attack_sound* =

**shoot* =

Задаем тип стрельбы. Возможные значения:

always

- значение по умолчанию, стреляет всегда, когда можно

none

- не стреляет вообще.

terminal

- стреляет только когда находится на последней точки патрульного пути. Это сделано для облегчения построения атакующих сцен.

ВНИМАНИЕ! У кемпера есть один большой минус – когда ему наносится хит и он не знает откуда хит наносится (не видит противника, не слышит выстрела), то он тупо продолжает стоять на старом месте и ждать следующей пули.

Ввиду этого не стоит расставлять кемперов в случае, когда сталкеры должны защищаться и держать позицию в том случае, если есть несколько направлений, откуда игрок или стелкеры смогут атаковать поставленного кемпера. Используйте *walk*еров в таких случаях, а кемперов стоять ставить для атак по путям и как снайперов.

В схеме читается стандартный сигнал:

path_end – NPC дошел до конечной точки пути

2.2.1. Схема sniper

Разновидность кемпера. Отличаются тем, что стреляют только одиночными выстрелами и не смотрят по точкам патрульного пути, а сканируют пространство между ними. Скорость сканирования от точки к точке фиксирована и равна 20сек.

NB! Ставить снайперу только 2 точки look

В секции кемпера прописать:

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

2.3. Схема remark

Схема используется для синхронизации\связки других схем или проигрывания анимации, реплик. Схему *remark* можно использовать и без задания параметров, в этом случае возьмутся параметры по умолчанию.

```
[remark]
anim = анимация ремарка, по умолчанию wait
target = направление, куда смотрит сталкер
```

<i>target = story actor or story_id</i>	– смотреть на игрока или на объект с заданным story_id
<i>target = path patrol_path, point_id</i>	– смотреть патрульный путь, где patrol_path – название пути, а point_id – номер точки пути
<i>target = job job_section, smart_name</i>	– смотреть на работу в заданном смарте, где job_section – имя работы, а smart_name – название смарт-террейна.
<i>target = nil</i>	– смотреть в никуда.

Пример:

target = vasya – персонаж будет смотреть на объект со story_id - vasya

Стандартные сигналы для remark:

<i>sound_end</i>	– по окончании проигрывания звуковой схемы
<i>anim_end</i>	– по окончании проигрывания анимации
<i>action_end</i>	– по окончании проигрывания и того и другого, если они синхронизированы

Пример синхронизации анимации и звука в схеме *remark*:

```
[remark]
anim = анимация
snd = звук
snd_anim_sync = true
on_signal = action_end | следующая схема
```

Поле anim_reset в схеме [remark] не работает.

2.4. Схема patrol

Схема для создания патруля. Представляет собой вариацию kamp только в состоянии ходьбы.

[patrol]

path_walk = *path_walk*

* *path_look* = *path_look* – не обязательный, только если в пути *path_walk* больше одной точки пути

* *formation* = *back*

* *commander* = *true* – назначить командиром, желательно, чтобы такой красивый он был один

* *move_type* = *patrol* – задает изначальный режим перемещения, по умолчанию *patrol*.

Анимации перечислены в файле *state_mgr_lib*

* *formation* = *back* – описывает способ построения и не является обязательным. Возможны следующие варианты:

back - мужики идут чуть позади командира в два ряда (по умолчанию)

line - шеренга

around - вокруг командира

При остановке командора в *meet* мужики останавливаются.

Если командир помирает, то автоматически будет выбран другой. Командиром становится тот, кто первый попал под схему. Способы построения можно задавать вейпоинтах следующим образом:

ret=0...2

0 – линия

1 – вокруг старшего

2 – по бокам

При движении командор работает как обычный *walker* и сопровождающие его кадры повторяют его действия. То есть, если в параметрах вейпоинта прописано *a=assault*, то командор помчится с оружием убийства на перевес, а остальные его откопируют.

2.5. Схема sleeper

Схема сидящего и спящего NPC. Необходимо поставить патрульный путь, минимум из 1 поинта. Спящий будет садиться спать в нулевой точке пути, и разворачиваться при этом в сторону первой точки.

[sleeper]

path_main = <имя пути>

* *wakeable* = *true* – может ли проснуться быстро (если *true*, то спит на корточках и во сне бормочет)

NB: Если путь состоит из двух точек, то связь нужно делать от первой точки к нулевой (либо двунаправленную).

2.6. Схема kamp

Схема сталкера, сидящего в определенном радиусе, вокруг указанной точки (у костра), и располагающегося лицом к этой точке.

[kamp]

center_point = *kamp_center* – имя точки вокруг которой NPC будет устраиваться.

`radius = 2` – насколько далеко сталкер будет сидеть от центра лагеря, 2- по умолчанию
`def_state_moving = run` – дефолтное состояние, в котором сталкер будет идти к точке кампа)

NB! Если точка кампа находится в костре, то в оффлайне сталкера придут на нее, а когда они перейдут в онлайн, то окажутся внутри костра, где и получают хит. Чтобы этого не случилось в секции кемпа указывать `path_walk` из одной точки, название которой = `<path_kamp_name>_task`

`path_walk = <path_kamp_name>_task`

Если точка кемпа расположена в чистом поле то, `path_walk` прописывать не надо.

2.7. Схема wounded (раненный)

Используется для настройки раненых.

`[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`

Значения полей:

`hp_state` – поведение персонажа когда он не видит игрока
`hp_state_see` – поведение персонажа, когда он видит игрока
`psy_state` – поведение персонажа при псиатаках
`hp_victim` – куда смотреть, в зависимости от ХП. Возможные значения: ***nil, actor, story_id***
`hp_cover` – идти в укрытие или нет, в зависимости от ХП
`hp_fight` – разрешено воевать или нет, в зависимости от ХП
`help_dialog` – идентификатор диалога вместо дефолтного `actor_help_wounded`. Если вам по сюжету необходимо заменить диалог другим, то вы в этом поле прописываете идентификатор другого диалога.
Также мы вставляем стартовый диалог раненого. Если мы его прописываем, то все актёрские диалоги для раненых должны иметь такой precondition: `dialogs.allow_wounded_dialog`.

Где:

`HP` – пороговое значение здоровья персонажа (от 0 до 1)
`PSY` – пороговые значения пси здоровья персонажа

Пример. В качестве примера взята дефолтная настройка.

`[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

```

Где:

best_pistol – проверка на то, что лучшее оружие НПС является пистолетом.

Пример 2 (NPC не будет падать раненым)

```

[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. Дополнительные настройки внутри схем

combat_ignore = *true* (NPC будет игнорировать бой)

combat_ignore_cond = {+info -info =func !func} *true* (NPC будет игнорировать бой по заданному кондлисту)

Функции, используемые для работы с кондлистом комбат игнора:

fighting_dist_ge(расстояние в метрах) –

универсальная функция для *combat_ignore*, проверка расстояния до врага

is_enemy_actor – текущий враг актёр?

enemy_in_zone(zone_name) – проверка находится ли враг в зоне *zone_name*

combat_ignore_keep_when_attacked = *true* (NPC продолжает игнорировать бой, даже если в него стреляет игрок)

out_restr=<*restricor_name*> (NPC не будет покидать указанный рестриктор)

in_restr = <*restricor_name*> (NPC не будет входить в указанный рестриктор)

invulnerable = *true* (делает персонажа неуязвимым)

show_spot = {+info1} *false* (NPC не будет отмечаться на радаре)

3. Настройки логики по событию.

3.1. Секция combat

Показывает, что происходит, когда NPC срыгается в бой.

Для задания различных типов скриптовых боёв для различных ситуаций используется параметр *combat_type*.

```

[logic]
active = walker
on_combat = combat (срабатывает, когда NPC вступает в бой)

```

[combat]

on_info = *%+info -info =func%* эффекты, которые вызываются на каждом раунде боя.

[walker]

combat_type = *{+info =func}* *camper* (monolith, zombied)

path_walk = ...

3.2. Секция **death**

Схема показывает, что происходит при смерти NPC.

on_death = *death*

[death]

on_info = *%+info -info =func%*

3.3. Секция **hit**

Схема показывает, что происходит при, нанесении повреждения NPC.

on_hit = *hit*

[hit]

on_info = *%+info -info =func%*

3.4. Секция **danger**

Настройка может задаваться только в какой-то схеме, например:

[walker]

danger = *danger_condition*

[danger_condition]

ignore_distance = 50 (расстояние указывается в метрах)

ignore_distance_grenade =

ignore_distance_corpse =

ignore_distance_hit =

ignore_distance_sound =

Можно также указывать время ожидания для денжера в зависимости от типа:
(дефолтные настройки)

danger_inertion_time_grenade = 20000

danger_inertion_time_corpse = 10000

danger_inertion_time_hit = 60000

danger_inertion_time_sound = 15000

Алгоритм работы такой: Сперва проверяется, что расстояние до опасности не отсекается по *ignore_danger*. Если опасность ближе, то тогда анализируется ее тип, и проверяется по соответствующему данному типу расстоянию. Если опасность ближе - тогда разрешается реакция на нее.

NB: если надо, чтобы в разных случаях сталкер игнорировал разные типы денжеров, создается несколько секций денжера:

[danger_condition@1]

ignore_distance = 50

[danger_condition@2]
ignore_distance = 20

* *danger_expiration_time* = Через сколько времени денжер перестанет быть актуальным.
Дефолт 5000 мс.

* *danger_inertion_time* = Через сколько времени персонаж забудет про денжер, на который он отреагировал. Дефолт 10000 мс.

3.5. Дополнительные секции

3.5.1. Секция *dont_spawn_character_supplies*

Если прописать эту секцию в кастом дату персонажу, то у него внутри не заспавнится стандартный набор барахла, прописанный в профиле.

[dont_spawn_character_supplies]

3.5.2. Секция *dont_spawn_loot*

Всякого рода сюжетные персонажи которые должны быть пустыми после смерти (например раненные или пленные) оказываются не пустыми. Чтобы это исправить необходимо в кастом дате персонажа прописать секцию

[dont_spawn_loot]

3.5.3. Секция *spawner*

Эта секция, которая присутствует как у NPC, так и у монстров, спавнит их по определенному условию (выводит в онлайн). Для того, чтобы они появились в данной точке, им надо поставить в настройках в Level editor флажок *no_move_in_offline* и отключен *can_switch_offline*. Спавнер прописывается в кастом дату объекта перед секцией *logic*

Работает *spawner* следующим образом:

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

Примечание. Если условия спавна не будет выполняться, то объект не заспавнится, а если он заспавнился и условие перестает выполняться, то объект будет спавнером уведен в оффлайн.

Пример:

[spawner]
cond = {=is_day}
(объект заспавнится днем и уйдет в оффлайн ночью)

После того, как объект заспавнился, его берет под управление скрипт *Logic*

3.5.4. Секция *known_info*

По обыску тела NPC, выдает прописанные в секции инфопоршни

[known_info]
info1
info2

4. Схемы логики для монстров

4.1. Схема mob_walker

Работает аналогично схеме обычного walker. Но есть некоторые отличия.

Флаги пути движения:

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

c=true - идти дальше в присяде; i=true - бежать дальше

sig=signal_name - установить заданный сигнал для xr_logic

Флаги пути обзора:

t=время_мсек - время в миллисекундах, которое нужно ждать, смотря в точку

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

В customdata персонажа задайте (* отмечены обязательные поля):

[walker]

path_walk = путь перемещения

path_look = путь обзора

*no_reset = true/false - не сбрасывать action предыдущей схемы (если нужно сохранить, например, звук). По умолчанию false.

*actor_friendly = true/false - монстр никогда первым не нападает на игрока, но если игрок хоть раз атакует монстра - этот режим навсегда отключится. По умолчанию false.

*npc_friendly = true/false - монстр никогда первым не нападет на другого монстра (даже враждебного).

*friendly = true/false - монстр не нападает ни на игрока, ни на монстров. В случае агрессии с их стороны, не запоминает их как врагов и остается дружелюбным ко всем. По умолчанию false.

Файл: \gamedata\scripts\mob_walker.script

У кровососов можно управлять невидимостью:

[mob_walker]

...

state = vis

или

state = invis

Задаст значение по умолчанию.

Также в флагах walk пути mob_walker-а можно использовать флажок b (behaviour) с теми же параметрами:

wp00|b=vis

wp00|b=invis

4.2. Схема mob_remark

Ремарковая схема, только не для сталкеров, а для монстров.

*state = специфическое состояние данного конкретного монстра (для кровососов - невидимость)

*dialog_cond = {+info, =func, -info, !func} условия для открытия окна диалога

*anim = анимации монстра, перечисляются через запятую.

*anim.head = анимации головы монстра, через запятую перечисляются

*tip = какой значок подсветится, при наведении на него курсора

**snd* = какой звук издает

**time* = время проигрывания анимаций, используется только для отладки.

4.3. Схема **mob_combat, mob_death**

Работают точно также как и у сталкеров соответствующие схемы.

4.4. Схема **mob_jump** (монстр-пружинка)

Схема **mob_jump**. Теперь **mob_jump** служит для задания прыжков монстров без каких либо проверок и ограничений (расстояние, углы и т.д.). Указывается позиция с помощью патрульного пути, смещение относительно этой позиции и физический фактор прыжка.

Пример:

```
[logic]
```

```
active = mob_jump
```

```
[mob_jump]
```

```
path_jump = path
```

```
ph_jump_factor = 2.8
```

```
offset = 0,10,0
```

```
on_signal = jumped | nil
```

path_jump – путь, с помощью которого мы задаем 1 целевую точку прыжка (с нулевым индексом). Реальная точка учитывает позицию **path_jump[0]** + смещение, заданное с помощью **offset**.

offset – смещение по осям x,y,z соответственно, с помощью которого задается реальная точка в пространстве (может не находится на аи-ноде).

ph_jump_factor - влияет на время прыжка. Визуально с помощью него задается кривизна траектории полёта. Чем он больше, тем прыжок более острый, быстрый (меньше дуга). С помощью данной схемы можно делать: перепрыгивание со здания на здание, выпрыгивание из окна, перепрыгивание высоких ограждений и др. Дефолтное значение = 1,8

Примечание:

Фактически **mob_jump** - это не состояние, а разовое действие. При переходе в него монстр разворачивается в сторону прыжка и прыгает, поднимая сигнал **jumped**. Т.е. "**on_signal = jumped | имя_схемы_или_nil**" – является обязательным параметром в схеме, чтобы знать куда переходить дальше.

При выборе позиции используется первая точка патрульного пути (0-вой индекс)

4.5. Схема **mob_home**

В этой схеме монстры будут ходить, или спать вокруг указанной точки.

Пример:

```
[mob_home]
```

```
path_home = path1
```

```
home_min_radius = 10
```

```
home_max_radius = 30
```

```
aggressive_home - в назначенную точку path_home монстры бегут а не идут.
```

Описание:

Монстры держатся вокруг точек пути `path_home`. В атаке бросаются на врага, если враг внутри `home_min` радиуса, иначе прячутся в укрытия. Отсюда следует, что `home_min` -радиус желательно делать таким, чтобы внутри было достаточно каверов. В айdle тоже обычно расходятся по каверам. `home_max` радиус сделан по принципу большого рестриктера в схеме «гнездо».

Добавлена возможность задания минимального и максимального радиусов для схемы `mob_home` в флагах первой точки пути (`path_home`). Для этого введены флаги `minr` и `maxr`. В случае, если радиусы заданы и в секции и во флагах, то значение радиуса берется из секции. Если не задано ни там, ни там, то берутся дефолтные значения 20 и 40 соответственно.

4.6. Дополнительные настройки внутри схем, для монстров:

<code>actor_friendly = true</code>	(монстр не атакует актера, до первой атаки на него)
<code>npc_friendly = true</code>	(монстр не атакует сталкеров и монстров, до первой атаки на него)
<code>friendly = true</code>	(то монстр не атакует никого до первой атаки на него)
<code>braindead = true</code>	(то монстр игнорирует любые атаки)

5. Настройка логики и переключение между схемами

В `customdata` любого персонажа (кроме свободных) должна присутствовать секция `[logic]`.

В секции должно присутствовать один из ключей:

`active` = активная схема, запускающаяся первой.

`cfg` = имя `ltx` файла с настройками

Если задано поле `cfg`, то в качестве настроек персонажа будет использовано содержимое указанного файла.

Не обязательные параметры:

`relation = neutral` - назначает отношение прс к игроку, *enemu* – враг, *neutral* – нейтральный, *friend* - дружественный

`sympathy = 0` - множитель влияния отношения прс к игроку, на отношения группировки к игроку

`level_spot =` - тип отметки и подписи на карте: **`quest_npc`** – *важный персонаж*, **`mechanic`** – *техник*, **`trader`** - *торговец*

`trade = misc\trade_generic.ltx` - назначается файл с настройками торговли

Пример. Настройки простого `walker-a`:

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

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

Переключение схем выполняется с помощью дополнительных условий схемы `logic`, которые прописываются в текущей схеме.

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

[camper]
```

path_walk = *walk2*
path_look = *look2*

Если *logic* переключает между несколькими одноименными схемами (например несколькими *walker*), то нужно через @ давать более информативные названия: *walker@day*, *walker@alarm* и т.д.

Условия для переключения схем:

on_info = {*func* +*info*} *walker* – по условию функции *func* и выдачи флажка *info*
on_timer = *1000* | *walker* – через 1с
on_game_timer = *10* | *walker* – через 1с
on_actor_dist_le = *10* | *walker* – когда игрок подойдет ближе 10м
on_actor_dist_le_nvis = *10* | *walker* – тоже самое, но без проверки на видимость
on_actor_dist_ge = *10* | *walker* – когда игрок будет на расстоянии больше 10м
on_actor_dist_ge_nvis = *10* | *walker* – тоже самое, но без проверки на видимость
on_signal = *signal* | *walker* – при получении сигнала (сигнал может быть получен из точки пути см. настройку путей, или один из стандартных сигналов)
path_end – NPC дошел до последней точки пути
sound_end – по окончанию проигрывания звука (см. настройки звуков)
on_actor_in_zone = *restrictor_name* | *scheme* – если актер в зоне (указывается имя рестриктора)
on_actor_not_in_zone = *restrictor_name* | *scheme* – если актер не в зоне (указывается имя рестриктора)
on_npc_in_zone = *story_id* | *restrictor_name* | *scheme* – если NPC, с заданным *story_id* в зоне (указывается *story_id* NPC, и имя рестриктора)
on_npc_not_in_zone = *npc_story_id* | *restrictor_name* | *scheme* – если NPC не в зоне (указывается *story_id* NPC, и имя рестриктора)
on_actor_inside = *scheme* – зона проверяет, находится ли игрок внутри нее (лучше используйте функцию *on_actor_in_zone*)
on_actor_outside = *scheme* – зона проверяет, находится ли игрок за ее пределами (лучше используйте функцию *on_actor_in_zone*)
on_offline = – условие перехода сталкеров в оффлайн, поддерживает кондлист

Можно указывать в любой секции логики, если указано в секции [*logic*...] то по дефолту будет браться из нее, если же указано с секции логики, то будет брать

NB: с любыми из вышеперечисленных параметров можно работать следующим образом:

on_info = {....} %...%
on_info2 = {....} %...%
on_info3 = {...} %...%
и так далее до посинения

Так же будет работать:

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

Единственное правило, все стоки должны быть разными, т.е. запись:

~~*on_info2*~~ = ...
on_info2 = ...

Работать не будет, сработает только второй переход.

NB: все параметры поддерживают кондлист:
on_timer = 1000 | {=func +info} walker

5.1. Настройки внутри секции

5.1. Настройка условий и запуск функций

Инфопоршн – условный флажок, для отметки событий. Имеет два состояния – «выдан» и «не выдан».

Кондлист – поле для проверки состояний инфопоршнов и возвращаемых функциями значений:
{+info =func} – проверка условий

%+info =func% – выдача инфопоршна, запуск функции

on_info = %+info1% – выдается инфопоршн *info1*
on_info = %-info1% – снимается инфопоршн *info1*
on_info = {+info1} – проверка, выдан ли инфопоршн *info1*
on_info = {-info1} – проверка, не выдан ли инфопоршн *info1*

В кондлистах можно делать проверки с помощью функций:

on_info = {=func} – проверка, что функция *func* возвратила **true**
on_info = {!func} – проверка, что функция *func* возвратила **false**
on_info = %=func% – запуск функции

5.2. Работа со звуками

Чтобы добавить новый звук нужно прописать его в файле **script_sound.ltx** либо в одном из включенных в него файлов.

В секции [list] прописать название звука, и создать секцию с тем же названием

```
[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= – тип звука, может быть **actor** – в голове у актера, **npc** – звук голоса NPC, **3d** – звук от объекта;
path= – путь к звуковому файлу (**ВНИМАНИЕ!** если тип звука **npc** то путь к файлу указывается от папки **characters_voice**)
Если указать не полное имя файла (*lvl_sound_name_*), то проигрываться будут все звуковые файлы, название которых начинается на указанное имя;
shuffle= **rnd** - повторяющийся звук, **seq** - однократный звук, **loop** - зацикленный звук

idle = – первые две цифры – задержка (в сек.) перед повторным проигрыванием звука, третья вероятность проигрывания

Для проигрывания звука в логике NPC или другого объекта, нужно воспользоваться функцией:

=play_sound(имя_звука)

Чтобы остановить:

=stop_sound(имя_звука)

В этих функциях работают сигналы:

sound_end – конец звука

theme_end – конец звуковой темы,

Если нужно проигрывать звук постоянно, нужно использовать функцию:

=play_sound_looped(имя_звука)

Чтобы остановить, зацикленный звук:

=stop_sound_looped(имя_звука)

Пример:

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

5.4. Счетчики

Есть возможность использовать счетчики для любых событий в игре (спаун объекта, убийство NPC и т.д.), и соответственно считывать их показания.

Функции, используемые для создания и задания значений счетчика:

=set_counter(имя_счетчика:число)	– создает счетчик с указанным значением;
=inc_counter(имя_счетчика:число)	– создает счетчик и добавляет значения, если нужно добавить «1», то число можно не указывать;
=dec_counter(имя_счетчика:число)	– создает счетчик и удаляет значения, если «1», то число можно не указывать;

Функции, используемые для проверки значений счетчика:

=counter_greater(имя_счетчика:число)	– возвращает true, если значение указанного счётчика больше указанного числа;
=counter_equal(имя_счетчика:число)	– возвращает true, если значение указанного счётчика равно указанному числу;

Пример:

```
[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. Катсцены (ролики на движке)

Управлять запуском камеры в катсценах, можно с помощью схемы рестриктора [sr_cutscene] либо с помощью функций:

=run_cam_effector(file_path\cam_effector_name:number:true>false) – запускает камероэффект от координат игрока
file_path – путь к файлу камероэффекта, прописывается от папки
 \gamedata\anim\camera_effects)
cam_effector_name – название файла камероэффекта
number – условный номер камероэффекта, нужен если камероэффект надо останавливать, необязательный параметр
true>false – зациклен или нет камероэффект

=run_cam_effector_global(cam_effector_name:number) – запускает камероэффект из глобальных координат (координаты хранятся в самом файле камероэффекта)

=stop_cam_effector(number) – останавливает камероэффект, с указанным номером

5.5. Постпроцессы

Для работы с постпроцессами можно использовать следующие функции:

=run_postprocess(имя_постпроцесса:number:true>false) – вызывает построцесс
number – условный номер построцесса, нужен если построцесс надо останавливать, необязательный параметр
true>false – зациклен или нет построцесс
=stop_postprocess(number) – останавливает постпроцесс, с указанным номером

Пример:

```

[logic]
active = sr_idle

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

```

В данном примере **agr_u_fade** название постпроцесса.

Готовые постпроцессы:

alcohol	– эффект опьянения
agr_u_fade	– затемнение экрана
agr_u_fade_water	– затемнение экрана с последующим помутнением
mar_fade	– затемнение экрана
mar_fade	– постепенное затемнение, а потом осветление экрана
blink	– белая вспышка

Полный список постпроцессов gamedata/anims

5.6. Числовые идентификаторы, story id

Любому спаун объекту можно присвоить идентификатор story_id. Чтобы это сделать, нужно прописать в файле spawn_sections_ следующую логику:

story_id = "имя_id" (где имя_id – уникальное имя)

Пример:

```
В файле spawn_sections_  
[zat_vasya]:stalker  
$spawn = "respawn\ zat_vasya"  
character_profile = zat_vasya  
story_id = zat_vasya
```

Или на уровне в секции logic спаун элемента:

```
[story_object]  
story_id = zat_vasya
```

ВНИМАНИЕ! Не должно быть двух идентификаторов с одинаковым названием.

5.7. Примеры логики

Пример: для того, чтобы персонаж ходил по пути walk1, а при приближении игрока на дистанцию 5 метров, переключался на путь walk2 (но только при условии, что он видит игрока), нужно написать следующее:

```
[logic]  
active = walker1  
  
[walker1]  
path_walk = walk1  
path_look = look1  
on_actor_dist_le = 5 | walker2  
  
[walker2]  
path_walk = walk2  
path_look = look2
```

Выше рассмотрено безусловное переключение секций. Перед именем секции в фигурных скобках {} можно задавать дополнительные условия, а после имени секции - так называемые "эффекты", которые заключить в знаки процента: %%. Эффекты будут применены только в случае активации секции. Можно не задавать имя секции, а задать только условия и/или эффекты. Тогда активной останется старая секция, но условия и эффекты будут все равно обработаны. Если все условия в фигурных скобках не выполняются, секция активирована не будет.

Пример:

```
on_actor_dist_le = 5 | {условие} walker2 %%эффекты%
```

Условия могут быть следующими:

+infoportion - требуется присутствие *infoportion* у actor
-infoportion - требуется отсутствие *infoportion* у actor
=func - требуется, чтобы *func* вернула true
!func - требуется, чтобы *func* вернулся false

Эффекты могут быть следующими:

+infoportion - в случае включения секции у actor будет установлен *infoportion*
-infoportion - в случае включения секции у actor будет убран *infoportion*
=func - в случае включения секции стартует функция *func*

Несколько условия или эффектов разделяются проблемами:

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

Можно задавать сразу несколько секций, разделенных запятыми. Порядок обхода при этом - слева направо. После срабатывания первого из условий, обход прекращается. В примере ниже, если установлен *info1*, будет включена схема *walker2*, иначе, если установлен *info2*, будет включена схема *walker3*, иначе будет включен *walker4*:

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

В описанном выше поле *active* секции *logic*, можно также задавать условия, например:

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

В логических условиях теперь принимается ключевое слово *never*, которое означает, что условие ложно. Например:

combat_ignore_cond = {=actor_enemy =actor_has_suit} always, {=actor_enemy} never
%...эффекты...%

Вышеприведенная конструкция включает игнорирование боя, если у NPC враг - игрок в костюме, но отключит его, если врагом является игрок, но без костюма, при этом сработают эффекты (%%) секции *never*. Таким образом, выбор секции *never* равносителен отсутствию секции (несрабатыванию условия), но эффекты в знаках процента при этом срабатывают.

Пример работы с секцией *nil*. Секция *nil* выводит из-под скриптовых схем персонажа, монстра или объект и отпускает его под управление движка. Это надо если какое-либо условие выполнившись 1 раз больше не нуждается в проверке, при этом экономятся ресурсы машины, которые на каждом апдейте проверяют это условие.

[logic]
active = sr_idle

[sr_idle]
on_actor_inside = nil %+esc_actor_inside%

То есть, при входе актера в рестриктор выдается инфопоршн и рестриктор уходит в секцию *nil*, больше не проверяя наличие игрока.

NB: Обратно из секции nil под скрипты объект вернуть уже невозможно! Учитывайте это, используя ее.

Вот пример достаточно сложной логики:

```
[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}
```

Рассмотрим ее пошагово. Вначале сталкер работает по схеме walker-а. При этом он игнорирует бой, пока не будет поставлен инфопоршн alert. Он ждет 25 секунд, после чего переходит в схему remark. В ремарке он проигрывает идловую анимацию, говорит на указанные темы, не поворачивается и не двигается и точно также игнорирует бой. Если по нему попадут (on_hit) или убьют (on_death), будет поставлен инфопоршн alert и он перестанет игнорировать бой (понятно, что если он будет трупом, то это ему не поможет, но их в сценке трое, и тогда сорвутся в бой все остальные). Если его убьют, то также будет поставлен инфопоршн trup3 который сообщит о том, что этот сталкер убит.

А вот логика его противника:

```
[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

```

Он идет в схеме walker, игнорируя бой (причем игнорируя в любой ситуации). Идет в составе группы assault_group. Когда он приходит в конечную точку маршрута (там он синхронизируется с остальными из группы, это приписано в путях) и получает сигнал assault, то переходит в схему camper. В этой схеме у него не прописан combat_ignore, поэтому он начинает стрелять по противнику. После того, как все трое противников будут убиты, каждый из них, умирая ставит инфопоршн trup1, trup2 или trup3 и когда все трое будут убиты, то он переключится на схему walker2 (подойдет к костру).

6. Схемы логики space_restrictor

Общее замечание: Чтобы исключить ситуацию, когда актёр проскакивает через рестриктор и тот не успевает сработать, старайтесь ставить рестриктор так, чтоб минимальная ширина была больше 2 метров.

6.1. Схема sr_idle

Предназначение данной схемы – включить другую схему при срабатывании одного из стандартных условий логической схемы.

Сама по себе схема ничего не делает.

Пример настроек рестриктора:

```

[logic]
active = sr_idle

[sr_idle]
on_actor_inside = nil %+esc_actor_inside%

```

Обратите внимание, что после срабатывания проверки активная схема переключается в nil, чтобы не продолжать бесполезную проверку на каждом апдейте. Можно не задавать nil.

Часто эта схема работает вместе со спавнером, рестриктор выдает инфопоршн, при входе в зону, а спавнер по нему уже кого-то спавнит.

6.2. Секция sr_no_weapon

Данная схема убирает оружие у игрока при входе в зону.

Пример настроек рестриктора:

```

[logic]
active = sr_no_weapon

[sr_no_weapon]

```

6.3. Секция `sr_light`

Зона, в которой фонарики у неписей будут включены независимо от времени суток.
Работает следующим образом:

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

```
[sr_light]  
light_on = true/false (свет включен/выключен)
```

Также работает вместе с кондлистом:

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

```
[sr_light]  
light_on = true/false (свет включен/выключен)  
on_info = {+info1} section %+info2%
```

6.4. Схема `sr_particle`

Данная система отыгрывает партиклы как статичные так и движущиеся в указанном месте и в указанное время. Работет она следующим образом:

1) для партикловой системы с путем камеры:

```
[sr_particle]  
name = explosions\campfire_03      -имя партикловой системы  
path = particle_test.anm           -имя пути камеры  
mode = 1                          (обязательно !!!)  
looped = true/false               -флаг заикленности партиклов
```

2) для партикловой системы с обычным патрульным путем:

```
[sr_particle]  
name = explosions\campfire_03      -имя партикловой системы  
path = part_points                 -имя патрульного пути  
mode = 2                          (обязательно !!!)  
looped = true/false               -флаг заикленности партиклов
```

В вейпоинтах можно задавать флаг `s=имя_звуковой_темы` и `d=число` время задержки перед проигрыванием (задается в миллисекундах. Если не задано, то 0). `s` - имя звуковой темы в `sound_themes.ph_snd_themes` из которой будет случайно выбран звук для проигрывания во время проигрывания партикла. Звук не заикливается и играет только один раз.. Результат = партиклы отыгрываются во всех вейпоинтах одновременно (или с задержкой см. выше).

При `looped=true` по окончании проигрывания партиклов, они будут запускаться сначала, но уже без задержек. Сигнал `particle_end` выдаваться не будет. При `looped=false` сигнал будет выдан, когда все источники партиклов отыграют.

Поддерживается кондлист. Если рестриктор переходит в другую секцию, то автоматически перестают отыгрываться партиклы и замолкают звуки при них. Этот рестриктор является объектом, отслеживающим партиклы и нет никакой необходимости чтобы игрок в него заходил.

6.5. Схема sr_timer

Пример использования:

```
[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%
```

Описания полей:

type - тип счетчика, инкрементирующий(inc) или декрементирующий(dec).

Если поле не задано - счетчик будет инкрементирующий

start_value - начальное значение счетчика в РЕАЛЬНЫХ миллисекундах. Для декрементирующих счетчиков задавать обязательно. Для инкрементирующих, если не задано, то считается с 0.

Переходы из секции **sr_timer** могут быть как по обычным условиям (**on_timer**, **on_info**) так и по специфическому условию **on_value**. В общем случае **on_value** Можно использовать для производства каких либо действий в зависимости от состояния счетчика. Например:

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

6.6. Схема sr_psy_antenna

Зоны с такой секцией позволяют управлять эффектами от пси-воздействия (на Янтаре и Радаре). Сейчас можно управлять интенсивностью излучения и интенсивностью получения повреждений.

Способ применения: Расставить зоны, в каждой зоне написать, сколько процентов к интенсивности излучения и повреждения она добавляет/отнимает. Зоны могут быть вложены друг в друга, пересекать друг друга.

eff_intensity = - увеличение/уменьшение в % от базового значения интенсивности излучения.
hit_intensity = - увеличение/уменьшение в % от базового значения наносимого повреждения.

Пример зоны, которая добавляет 70% излучения:

```
[logic]
active = sr_psy_antenna
```

```
[sr_psy_antenna]
eff_intensity = 70
hit_intensity = 70
```

Пример зоны, которая убирает 30% излучения:

```
[logic]
```

```
active = sr_psy_antenna
```

```
[sr_psy_antenna]  
intensity = -30
```

6.7. Схема sr_cutscene

В нашем движке можно воспользоваться камерой (катсцена):

Пример:

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

Здесь мы проверяем инфопоршен *agru_nvidia_presentation* и функцию *black_screen* (нет ли черного экрана) и переходим в секцию *sr_cutscene@cam1*

```
[sr_cutscene@cam1]  
point = agru_nv_camera_walk      – точка walk где игрок будет находится после камеры  
  
look = agru_nv_camera_look      – точка look куда будет направлена камера игрока после камеры  
  
cam_effector = scenario_cam\Agraprom_underground\camera1_0_904 – файл камеры и путь к нему  
  
on_signal = cameff_end | sr_cutscene@cam2      – сигнал окончание камеры и переход в следующую секцию  
  
global_cameffect = true/false      – флажок отыгрывания камеры от глобальных координат уровня, а не от камеры игрока (по умолчанию false)  
  
[sr_cutscene@cam2]
```

Внимание! Камера от рестриктора будет работать лишь в случае, когда

- 1) на момент ее запуска игрок находится внутри данного рестриктора
- 2) рестриктор имеет тип **NONE default restrictor** в левел эдиторе

7. Набор дополнительных настроек логики у разных объектов.

Для всех физических объектов есть секция *ph_idle*, поддерживающая кондлист в которую можно при необходимости переводить объекты.

7.1. Схема работы двери, секция *ph_door*

NB! Для двухстворчатых ворот задается все аналогично.

locked = false\true Заперта ли дверь. По дефолту – false.

closed = false\true Закрыта ли дверь. По дефолту - true

tip_open = (если *locked == false*, то *tip_door_open*, иначе *tip_door_locked*)

Подсказка, которая появляется около прицела при наведении на дверь, если дверь закрыта.

tip_close = (если *locked == false*, то *tip_door_close*, иначе пустое значение)

Подсказка, которая появляется около прицела при наведении на дверь, если дверь открыта.

snd_init = Звук, который будет отыгран сразу при включении схемы.

snd_open_start = Звук, который будет отыгран при попытке открыть дверь.

snd_close_start = Звук, который будет отыгран при попытке закрыть дверь.

snd_close_stop = Звук, который будет отыгран, когда дверь захлопнется до конца.

Примеры:

Если нужно сделать дверь, которая при каком-то событии открывается со щелчком, то можно воспользоваться полем *snd_init* и переключением схем. В примере ниже при включении схемы *ph_door@unlocked* проиграется *snd_init*, т.е. *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. Схема работы кнопки, секция *ph_button*

При нажатии на кнопку переключает секции и выдает инфопоршн.

[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 – что происходит при нажатии

anim – анимация, которая отигрывается при нажатии на кнопку
anim_blend – плавная, сглаженная анимация. Может принимать значения true/false

***tooltip** – предназначено для того, чтобы задавать текстовую подсказку при наведении на кнопку. Текстовая подсказка нужна для того, чтобы как минимум было понятно, что этот девайс можно нажимать.

Пример настройки кнопки:

```
[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
```

Для того чтобы сообщение не потеряло адекватность при различных настройках клавиатуры сообщение следует писать с использованием токенов. Например:

```
<string id="tips_labx16switcher_press">  
  <text>Чтобы отключить чудо установку нажмите ($$ACTION_USE$$)</text>  
</string>
```

Вот пример кнопки, которая срабатывает не всегда, а по определенному условию:

```
[logic]
```

```
active = ph_button@locked
```

```
[ph_button@locked]
```

```
anim = button_false – анимация несрабатывания кнопки
```

```
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. Схема работы ворот, секция ph_gate:

То же самое, что и ph_door, но для ворот, состоящих из двух дверей:

Вместо параметров closed и locked сейчас используются параметры:

state: состояние, в котором дверь находится при инициализации (по умолчанию none)

open - в открытом

closed - в закрытом

none - в текущем (дефолтном или оставшемся от предыдущей схемы)

locking: блокировка дверей (по умолчанию none)

stick - прилипание дверей к крайним состояниям (пока в процессе настройки)

soft - дверь заблокирована с помощью силы, т.е. можно ее открыть/пробить машиной

Состояния в этом положении:

open - блокировать в открытом состоянии

closed - в закрытом

none - не используется (мягкая блокировка возможна только в крайних положениях)

hard - блокировка двери с помощью границ. Ворота можно только сломать

Состояния в этом положении:

open - блокировать в открытом состоянии

closed - в закрытом

none - в текущем

none - дверь не заблокирована

Общие параметры:

left_limit, right_limit - задают угол [0-180] открытия каждой из створок ворот. По умолчанию - 100 градусов.

breakable - (true/false) определяет можно ли сломать ворота. По умолчанию true.

Звуковые параметры аналогичны ph_door

Примеры:

[ph_gate@locked] ;блокировка в открытом состоянии, неразбиваемые.

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. Кодовые замки, ph_code:

При введении указанного кода выдает инфопоршн

[logic]

active = ph_code@lock

[ph_code@lock]

code = 1243

on_code = %+infoportion%

7.5. Толкнуть физический объект, ph_force

Схема позволяет пнуть предмет в указанную сторону. Прописывается в кастом дате предмета.

force = сила, которая прикладывается к объекту. Измеряется в убитых енотах

time = время прикладывания силы к предмету (в секундах)
**delay* = задержка (в секундах) перед применением силы
point = имя патрульного пути, точки которого будут использованы как цели (куда направлять предмет)
point_index = индекс точки патрульного пути, в стону которого полетит предмет.

7.6. Запретить швырять объект, **ph_heavy**

Прописывается в физ объектах, которые запрещены для швыряния бюрерам и полтергейстам. Например, если они должны лежать на конкретном месте (типа документов сюжетных) или слишком громоздки по габаритам, чтобы их можно было красиво кидать.
В кастом дате пишем:

[ph_heavy]

7.7. Раскачивание физики, **ph_oscillate**

Схема предназначена для плавного раскачивания физики (лампы, висющие зомби и т.д.)
Пример логики

[ph_oscillate]
joint = provod – имя кости к которой будет применена сила
force = 5 – собственно сила (в ньютонах)
period = 1000 – время прикладывания силы.

Сила прикладывается к кости объекта с линейным нарастанием. То есть в течении заданного периода времени сила вырастет с 0 до заявленного значения. После этого настанет пауза (сила не применяется) на время *period/2*. После окончания паузы сила применяется так же, как и в начале, но в обратном направлении.

7.8. Реакция на звук

Можно сделать, чтобы НПС или монстр реагировали на громкость определенного звука.
on_sound = story_id | sound_type | distance | sound_power | {conditions} section %effects%

Типы звуков:

sound_type:

WPN_hit – звук попадания пули
WPN_reload – звук перезарядки оружия
WPN_empty – звук попытки выстрелить из незаряженного оружия
WPN_shoot – звук выстрела оружия
MST_die – звук смерти
MST_damage – звук получения урона
MST_step – звук шагов

Переход состоится если объект услышит звук(**sound_type**) от объекта(**story_id**) при дистанции \leq **distance** и силе звука \geq **sound_power**.

Также поддерживается запись:

on_sound1 =
on_sound2 =

Если нужно отловить звук с любой силой, то **sound_power** указываем 0.
Если нужно отловить звук на любой дистанции, то **distance** ставим 10000.

Пример:

[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 ;Услышит все, кроме пистолета с глушителем

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 ;Примерно с 3х метров услышит

on_sound7 = zat_cop_id|WPN_reload|10|0.3| mob_home@fight ;Примерно с 3х метров услышит

8. Управление некоторыми фичами

8.1. Настройка реакции NPC, meet_manager

Синтаксис:

[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 ;Стартовая озвучка ,если видит игрока

sound_start_wpn = nil ;Стартовая озвучка ,если видит игрока с оружием

sound_stop = nil ;Озвучка, когда игрок стоит на месте

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 ;По умолчанию true. Можно ли торговать с NPC
<i>allow_break</i>	= true/false ;По умолчанию false. Можно ли выйти из диалога
<i>quest_npc</i>	= true/false ;По умолчанию false. Квестовый ли NPC

Вся настройка встречи отныне будет производится в отдельной секции. В секции logic или в текущей схеме можно будет указать, какую именно секцию с настройкой нужно использовать. Секция, которая указана в секции logic будет влиять на обработку встречи свободногуляющим сталкером.

Перечень полей:

meet_state, *meet_state_wpn* – задает анимацию и озвучку персонажа, в зависимости от расстояния до актера. Для случая если актер безоружен либо вооружен соответственно.

victim, *victim_wpn* – задает объект, на который должен будет смотреть персонаж. Возможные параметры: *nil* – никуда не смотрит, *actor* – смотрит на игрока, *story_id* – номер стори айди персонажа, на которого нужно будет смотреть.

use, *use_wpn* – настройки юзабельности персонажа. Возможны три варианта: *true*, *false*, *self*. При *self* НПС сам юзнет игрока, как только сможет дотянуться □

zone – Содержит набор имен рестрикторов, а также анимаций и озвучки, которую НПС будет отыгрывать, если игрок будет замечен в рестрикторе

meet_dialog – стартовый диалог НПС.

synpairs – содержит набор пар состояние_тела@звуковая_тема. Если при каком то наборе условий встреча будет отыгрывать именно это состояние и эту звуковую тему – то они будут синхронизироваться по рандомным анимациям состояния тела.

abuse – по умолчанию *true*, если *false*, то неюзающийся противник не будет обижаться.

Любую строку(в общей схеме они написаны строчными буквами) можно задавать кондлистом.
({+info1 –info2} ward %+info%) *quest_npc* – по умолчанию

Для облегчения настройки встречи сделана возможность упрощенного задания дефолта:

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

Саму секцию [default_meet] задавать не надо. Все настройки и так возьмутся из дефолта. Теперь о том, как с помощью этого конструктора собрать ту реакцию на актера, которая вам нужна (Во всех примерах зеленым цветом выделены состояния *state_manager*, синим – звуковые темы):

Ситуация 1

Игрок вдалеке подзывает нас рукой, при приближении просит убрать оружие, потом согласен говорить.

```
[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
```

Ситуация 2

Сталкер завидя нас просит убрать оружие. После этого подходит и заговаривает с нами. Если мы начинаем уходить от него или достаем оружие – начинает нас стрелять.

```

[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

```

Здесь: info – инфоропшн, который указывает что мы уже опустили оружие и были достаточно близко к НПС

Info2 – инфопоршн, который устанавливается в диалоге и говорит что персонаж уже сказал нам все, что хотел.

Killactor – функция в хг_effects которая обижает НПС на игрока.

Ситуация 3

Персонаж ходит по патрульному пути на заставе лагеря. Если игрок имеет допуск в лагерь – пропускает его и здоровается, иначе сперва отпугивает, а если игрок пробрался в лагерь – то обижается на него. При этом диалог зависит от того, имеет игрок допуск в лагерь или нет.

```

[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

```

Здесь:

True – вместо анимации, атаковать игрока.

Info – Инфопоршн, который говорит что мы имеем допуск к лагерю

Warnzone – рестриктор, в котором нас предупреждают

Kampzone – рестриктор, в котором нас убивают

Dialog1 – стартовый диалог НПС, если мы имеем допуск в лагерь

Dialog2 – стартовый диалог НПС, если мы не имеем допуск в лагерь.

Дефолтные настройки:

По дефолту встреча настроена со следующими параметрами:

```

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

```

syndata

= *hello@hail|backoff@threat_weap*

NB: Если нужно, чтобы сталкер не разговаривал с игроком в данной секции, необходимо прописать ему:

meet = no_meet

8.2. Смарткаверы

Смарткаверы - это спаун объект, который управляет анимациями NPC. Используется для создания сценок со сложной анимацией. Тип смарткавера (анимации которые может использовать NPC в смарткавере), зависят от параметра **description** (выбирается в Level editor).

Боевые смарткаверы, управляют боевыми анимациями, их могут использовать NPC в универсальной боевой схеме. Так же, боевые смарткаверы могут использоваться для построения скриптовых сцен. Набор аниамций используемый NPC в смарткавере, зависит от выбранной бойницы (лупхолы). NPC может вести огонь из смарткавера только если противник находится внутри радиуса действия лупхолы. Названия лупхол для боевых смарткаверов смотреть в файле:

Battle loopholes.xls

Анимационные смарткаверы, настраиваются аналогично боевым, но используют специальные анимации, используются только для скриптовых сцен. Названия лупхол для анимационных смарткаверов смотреть в файле: **Lead loopholes.xls**

<i>[smartcover]</i>	
<i>cover_name</i> = <i>cover_1</i>	– имя смарткавера, обязательный параметр
<i>loophole_name</i> = <i>lh1</i>	– имя лупхолы, какие лупхолы доступны, зависит от типа смарткавера (в боевых смарткаверах, имя лупхолы)
<i>cover_state</i> = <i>fire_target</i>	– состояние NPC смарткавере: <i>fire_target</i> – стреляет по цели, <i>fire_no_lookout_target</i> – стреляет по цели не высываясь, <i>idle_target</i> – спрятался, <i>lookout_target</i> – выглядывает, <i>default_behaviour</i> – прячется, выглядывает, если есть цель, атакует
<i>use_in_combat</i> = <i>true</i>	– использование смарткавера в бою, если не поставить этот параметр NPC будет срываться в универсальную боевую схему
<i>target_enemy</i> =	– цель, куда стрелять: <i>actor</i> или <i>story_id</i>
<i>target_path</i> = <i><path_name></i>	– будет стрелять в первую точку указанного пути
<i>idle_min_time</i>	– настройка таймингов между анимациями
<i>idle_max_time</i>	– настройка таймингов между анимациями
<i>lookout_min_time</i>	– настройка таймингов между анимациями
<i>lookout_max_time</i>	– настройка таймингов между анимациями

ВНИМАНИЕ! Перед тем как прописать в смарткавере лупхолу, нужно убедиться что лупхола с таким названием есть в этом смарткавере (см. в файлах **Battle loopholes.xls**, **Lead loopholes.xls**).

Смарткаверы используют стандартные сигналы:

<i>enemy_in_fov</i>	– противник находится в лупhole
<i>enemy_not_in_fov</i>	– противник не находится в лупhole

8.3. Вывод текста на экран

Для того, чтобы отметить что-то срочное и важное. Например, срочное сообщение о том, что надо покинуть зону, можно использовать текст. Он будет находиться выше перекрестия.

Пример добавление текста:

```
[logic]
active = sr_idle
[sr_idle]
on_info = %=add_cs_text(agru_cs_warning)%
; Где agru_cs_warning = id строки
```

Пример удаления текста:

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

8.4. Включение и выключение аномалий

Некоторые специфические аномалии (например **agru_firetude**) можно включать и выключать функциями **=disable_anomaly(game_story_id аномалии)** и **=enable_anomaly(game_story_id аномалии)**.

Внимание! По умолчанию аномалия включена.

Выключение аномалии:

```
[logic]
active = sr_idle

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

Включения аномалии:

```
[logic]
active = sr_idle

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

8.5. Работа с текстами

Все тексты создаются в XML файлах, которые находятся в
\\gamedata\\configs\\text\\папка_локализации

В файле XML

```
<string id="lvl_item_name">
    <text>Текст</text>
</string>
```

Идентификатор текста нужно называть по правилам:

Первым пишется сокращенное название уровня («**agr**» – agroprom), в случаях, если текст не относится к определенному уровню, следует добавлять **st_**

Потом пишется сокращенное название сцены, персонажа, или предмета.

Последним пишется, чем является этот текст:

name – название

text – текст задания

descr – подробное описание

8.6. Отключение посткомбата

Можно отключить у сталкеров посткомбат (ожидание после боя). Для этого нужно в секции **[logic]** прописать ключ **post_combat_time** со значением «0, 0»

```
[logic]
```

```
active = walker
```

```
post_combat_time = 0, 0
```

8.6. Настройки физических объектов

[collide] - секция для настройки коллизии объекта;

ignore_static - отключение коллизии со статикой;

small_object - объект считается **small_object**, нужно для настройки коллизии с физикой

ignore_small_objects

9. Синтаксис файлов LTX

Синтаксис файлов **LTX** повторяет синтаксис файлов **INI**

9.1. Секции, ключи

```
[section]
```

```
key = value1
```

9.2. Наследования

Секции в файлах **LTX** можно наследовать от других секций, для этого после закрывающей скобки нужно поставить двоеточие и перечислить, через запятую, названия секций которые нужно наследовать. При наследовании, все ключи со значениями, переходят из наследуемой секции в секцию наследник. Секция наследник должна находиться ниже от наследуемой секции.

Пример:

```
[section_1]
```

```
key1 = 1
```

```
key2 = 2
```

```
key3 = 3
```

```
[section_2]: section_1
```

[section_3]: section_1, section_2

9.3. Инклюды

В файл LTX можно вложить другой LTX файл.

#include "file_path\included_file.ltx"

Путь указывается от папки с файлом, в который делается вложение.