Настройка логики (Зов Припяти)

Материал из xrWiki

Содержание

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 найти 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 прибудут все персонажи данной **team** (**team** задается в виде текстовой строки в **customdata**). До тех пор, пока остальные персонажи не прибудут, ожидающей персонаж будет отыгрывать свою idle-анимацию.
- **sigtm=signal**
  Устанавливает сигнал при вызове **time_callback**-а **state manager**-ом. Соответственно, если **t=0**, то сигнал будет установлен после отыгрывания 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 так:
  wp00|a=press|t=0|sigtm=pressed
  А затем переключить схему:
  on_signal = pressed | другая_схема

Более подробное описание путей

Walker

Файл:logic-setup-walker.png
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 = ...**
  Имя пути, описанного в п. 2 (необязательно). 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:

КАРТИНКА1

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.

Заметка

В точках пути **path_walk** используются анимации только из раздела «Ходячие состояния»!

Пример 2

Чтобы персонаж говорил, перемещаясь по маршруту, нужно определить в каждой точке список тем, на которые он может говорить. Для этого существует поле **s = имя_звуковой_схемы** (по умолчанию звук отключен). Несколько тем можно перечислять через запятую.

Пример 3

КАРТИНКА 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 (**sa**, **sf**, **sp**, **st**) задавать не рекомендуется, значения по умолчанию приемлемы для большинства скриптов. Если нужно стартовать звук одновременно с анимацией, лучше воспользоваться полями пути **path_look**, о котором будет написано ниже. 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:

КАРТИНКА 4

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 случайным образом выберет одну из подходящих точек.

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

- **p = ...** — 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, и 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).
  Если мы напишем **a=hide**, то персонаж сядет в состоянии **danger**, если **a=guard**, то встанет с оружием наизготовку и т.д.

Заметка

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

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.

Схема walker

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

***Серым** цветом выделены необязательные параметры.*

[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. Необязательный, 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 = ...**
  Стандартное состояние, в котором он стоит и смотрит на точку, если в этой точке не задано другое состояние (guard по умолчанию).

В схеме читается стандартный сигнал:

- **path_end** — NPC дошел до конечной точки пути.

Схема camper

Свойства кемперов:

1. Кемпер стоит на точке и смотрит в направлении, куда Вы его поставили в редакторе или передвигается по патрульным путям.
2. Кемперы переключаются на универсальный комбат, только если видят врага ближе чем в 30 метрах. Если он выжил, он возращается в состояние кемпера. В любых других случаях действуют по собственной скриптовой схеме. Если видим врага — стреляем. Если слышим дэнжер — то смотрим в направление в дэнжере. Если видим гранату — убегаем от гранаты. Если видели врага, а враг исчез, то смотрим в точку, где видели его последний раз.
3. Кемперы не сражаются в движении. Если они видят врага, то останавливаются, стреляют, а потом продолжают движение.

[camper]

- **path_look = patrol_path**
  Необязательный, только если в пути **path_walk** больше одной точки пути.
- **radius = ...**
  Расстояние в метрах. Если расстояние между кэмпером и противником меньше указанного, кемпер 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. Ставите 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 виде врага (по умолчанию sneak).
- **def_state_moving_fire = ...**
  Состояние (анимация), в котором сталкер отстреливается от врага во время движения на ближайшую точку пути (sneak_fie).
- **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 умолчанию звуковая тема "fight_attack". Можно изменить на любое другое (для сценических потребностей) либо вообще отключить, поставив пустое значение.
- **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 точке патрульного пути. Это сделано для облегчения построение 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 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 в таких случаях walker-ов, а кемперов стоит ставить для атак по путям и как снайперов.

В схеме читается стандартный сигнал:

- **path_end** — NPC дошел до конечной точки пути.

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 20 сек.

Заметка

Ставьте снайперу только 2 точки **look**.

В секции кемпера прописать:

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

Схема remark

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

[remark]

- **anim = ...**
  Анимация ремарка, по умолчанию **wait**.
- **target = ...**
  Направление, куда смотрит сталкер.
- **target = story | actor or story_id**
  Смотреть на игрока или на объект с заданным **story_id**.
- **target = path | patrol_path, point_id**
  Смотреть патрульный путь, где patrol_path — название пути, а **point_id** – номер точки пути.
- **target = job | job_section, smart_name**
  Смотреть на работу в заданном смарте, где **job_section** — имя работы, а smart_name — название смарт-террейна.
- **target = nil**
  Смотреть в никуда.

Стандартные сигналы для **remark**:

- **sound_end** — по окончании проигрывания звуковой схемы
- **anim_end** — по окончании проигрывания анимации
- **action_end** — 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
```

Заметка

Поле **anim_reset** в схеме **[remark]** не работает.

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. По умолчанию 2 м.
- **def_state_moving = run**
  Состояние по умолчанию, в котором сталкер будет идти к точке kamp-а

Заметка

Если точка кампа находится в костре, то в оффлайне сталкера прийдут на нее, а когда они перейдут в онлайн, то окажутся внутри костра, где и получат хит. Чтобы этого не случалось, в секции kamp-а нужно указывать **path_walk** из одной точки:
• **path_walk = <path_kamp_name>_task**

Если точка **kamp**-а расположена в чистом поле, то **path_walk** прописывать не надо.

Схема patrol

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

[patrol]

- **path_walk = path_walk**
- **path_look = path_look**
  Необязательный, только если в пути **path_walk** больше одной точки пути.
- **formation = back**
  Описывет способ построения. Возможны следующие варианты:
  - back - мужики идут чуть позади командира в два ряда (по умолчанию)
  - line - шеренга
  - around - вокруг командира
- **commander = true**
  Назначить командиром. Желательно, чтобы такой красивый он был один.
- **move_type = patrol** – задает изначальный режим перемещения, по умолчанию patrol.
  Анимации перечислены в *gamedata\scripts\state_mgr_lib.script*.

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 в точках пути следующим образом:

- **ret=0..2**, где
  - **0** — линия
  - **1** — вокруг старшего
  - **2** — по бокам

При движении командор работает как обычный **walker**, а сопровождающие кадры повторяют

его действия. То есть, если в параметрах точки пути прописано **a=assault**, то командор помчится с орудием убийства на перевес, а остальные его откопируют.

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).

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

Схема wounded

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 пси-атаках.
- **hp_victim**
  Куда смотреть, в зависимости от уровня здоровья. Возможные значения:
  - **nil**
  - **actor**

    - **story_id**
- **hp_cover**
  Идти в укрытие или нет, в зависимости от уровня здоровья.
- **hp_fight**
  Разрешено воевать или нет, в зависимости от уровня здоровья.
- **help_dialog**
  Идентификатор диалога вместо стандартного **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
```

- **HP** — пороговые значение здоровья персонажа (от 0 до 1)
- **PSY** — пороговые значения пси-здоровья персонажа

Пример 1

```
![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,wounde
d_psy
hp_victim = 30|actor|10|nil
hp_cover = 30|true|10|false
hp_fight = 30|true|10|false
syndata = wounded@help
```

- **best_pistol** — проверка на то, что лучшее оружие НПС является пистолетом.

Пример 2

```
![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
```

Здесь NPC не будет падать раненым.

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 **combat_ignore**:
  - **fighting_dist_ge(расстояние_в_метрах)**
    универсальная функция для **combat_ignore**, проверка расстояния до врага
  - **is_enemy_actor**
    текущий враг — actor?
  - **enemy_in_zone(zone_name)**
    находится ли враг в **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**
  Неуязвимость персонажа.
- **show_spot = {+info1} false**
  NPC не будет отмечаться на радаре.

Настройки логики по событию

Секция combat

Показывает, что происходит, когда NPC срывается 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 = …
```

Секция death

Схема показывает, что происходит при смерти NPC.

```
on_death = death

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

Секция hit

Схема показывает, что происходит при нанесении повреждения персонажу.

```
on_hit = hit

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

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 (настройки по умолчанию):

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

Алгоритм работы следующий. Сперва проверяется, что расстояние до опасности не отсекается по **ignore_danger**. Если опасность ближе, то тогда анализируется её тип, и проверяется по соответствующему данному типу расстоянию. Если опасность ближе — тогда разрешается реакция на неё.

Заметка

Если надо, чтобы в разных случаях сталкер игнорировал разные типы опасности, создаётся несколько секций ganger-а:

```
[danger_condition@1]
ignore_distance = 50
[danger_condition@2]
ignore_distance = 20
```

- **danger_expiration_time = …**
  Через какое количество времени опасность перестанет быть актуальной (по умолчанию 5000 мс).
- **danger_inertion_time = …**

Через какое количество времени персонаж забудет про опасность, на которую он отреагировал (по умолчанию 10000 мс).

Дополнительные секции

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]**

Секция dont_spawn_loot

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

- **[dont_spawn_loot]**

Секция spawner

Эта секция, присутствующая 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 LevelEditor флажок **no_move_in_offline** и отключить **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 перестаёт 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**.

Секция known_info

По обыску тела NPC выдает прописанные в секции инфопорции.

```
[known_info]
info1
info2
```

Схемы логики для монстров

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 или
- **r=true**
  бежать.
- **sig=signal_name**
  Установить заданный сигнал для **xr_logic**.

Флаги пути обзора:

- **t=время**
  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 **custom_data** персонажа задайте:
[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, то этот режим навсегда отключится. По умолчанию **false**.
- **npc_friendly = true/false**
  Монстр никогда первым не нападёт 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 дружественным ко всем. По умолчанию **false**.

Файл: `gamedata\scripts\mob_walker.script`

У кровосососов можно управлять невидимостью:
[mob_walker]
...
state = vis или **state = invis** задаёт 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**

Схема mob_remark

Ремарковая схема 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, перечисляются через запятую.
- **tip = ...**
  Какой значок подсветится при наведении на него курсора.
- **snd = ...**
  Какой звук издаёт.
- **time = ...**
  Время проигрывания анимаций, используется только для отладки.

Схемы mob_combat, mob_death

Работают точно так же, как и соответствующие схемы сталкеров.

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.). Указываются 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 первую 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 ИИ-ноде).

- **ph_jump_factor**
  Влияет на время прыжка. Визуально с помощью него задаётся кривизна траектории полёта. Чем фактор 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. Значение по умолчанию равно **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 (нулевой индекс).

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.
```

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 радиус **home_min** желательно делать таким, чтобы внутри было достаточно cover-ов. В idle тоже обычно расходятся по укрытиям. Радиус **home_max** сделан по принципу большого рестриктора 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.

Дополнительные настройки внутри схем для монстров

- **actor_friendly = true**
  Монстр не атакует актера до первой атаки на него.
- **npc_friendly = true**
  Монстр не атакует сталкеров и монстров до первой атаки на него.
- **friendly = true**
  Монстр не атакует никого до первой атаки на него.

- **braindead = true**
  Монстр игнорирует любые атаки.

Настройка логики и переключение между схемами

Секция logic

В **custom_data** любого персонажа (кроме свободных) должна присутствовать секция **[logic]**.

В секции должен 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.

Необязательные параметры:

- **relation = neutral**
  Назначает отношение NPC к игроку:
  - **enemy** — враг;
  - **neutral** — нейтральный;
  - **friend** — дружественный.
- **sympathy = 0**
  Множитель влияния отношения *NPC* 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**
  Файл с настройками торговли.

Пример (настройки простого **walker**-а):

```
[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 1 с.
- **on_game_timer = 10 | walker**
  через 1 с.
- **on_actor_dist_le = 10 | walker**
  когда игрок подойдет ближе 10 м.
- **on_actor_dist_le_nvis = 10 | walker**
  то же самое, но без проверки на видимость
- **on_actor_dist_ge = 10 | walker**
  когда игрок будет на расстоянии больше 10 м.
- **on_actor_dist_ge_nvis = 10 | walker**
  то же самое, но без проверки на видимость
- **on_signal = signal | walker**
  при получении сигнала (сигнал может быть получен из точки пути (см. настройку путей) или быть одним 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 неё (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 умолчанию

будет браться из неё.

Заметка

С любыми из вышеперечисленных параметров можно работать следующим образом:

```
on_info = {…} %…%
on_info2 = {…} %…%
on_info3 = {…} %…%
```

И так далее.

Точно так же будет работать следующий код:

```
on_info9 = {…} %…%
on_info7 = {…} %…%
on_info5 = {…} %…%
```

Единственное правило — все строки должны иметь разные имена, т.е. запись:

```
on_info2 = {…}
on_info2 = {…}
```

работать не будет, сработает только второй переход.

Заметка

Все параметры поддерживают кондлист:

```
on_timer = 1000 | {=func +info} walker
```

Настройка условий и запуск функций

Инфопоршн (infoportion) — условный флажок для отметки игровых 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
- **on_info = {+info1}**
  проверка, выдан ли инфопоршн
- **on_info = {-info1}**
  проверка, не выдан ли инфопоршн

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

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:
  - **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 неполное имя файла (`lvl_sound_name_`), то проигрываться будут все звуковые файлы, название которых начинается на указанное имя.
- **shuffle = …**
  Задает режим проигрывания:
  - rnd — случайный повторяющийся звук
  - seq — одноразовый 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, используйте 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
```

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
```

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\anims\camera_effects`.
  - **cam_effector_name**

Имя файла камероэффекта.
  - **number**
    Условный номер камероэффекта. Нужен, если камероэффект надо останавливать.
  - **true\false**
    Зациклен камероэффект или нет.
- **=run_cam_effector_global(cam_effector_name:number)**
  Запускает камероэффект из глобальных координат (координаты хранятся в самом файле камероээфекта).
- **=stop_cam_effector(number)**
  Останавливает камероэффект с указанным номером.

Постпроцессы

Для работы с постпроцессами можно использовать функции, перечисленные ниже.

- **=run_postprocess(имя_постпроцесса:number:true\false)**
  Вызывает постпроцесс.
  - **number**
    Условный номер построцесса. Нужен, если построцесс надо останавливать.
  - **true\false**
    Зациклен построцесс или нет.
- **=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.

Стандартные 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 см. в `gamedata/anims`

Числовые идентификаторы (story id)

Любому спаун-объекту можно присвоить индификатор **story_id**. Чтобы это сделать, нужно прописать в файле `spawn_sections_*.ltx` логику такого вида:

- **story_id = "уникальное_имя_id"**

Пример (spawn_sections_*.ltx):

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

Или на уровне в секции `logic` спавн-элемента:

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

Заметка

Не должно быть двух идентификаторов с одинаковыми именами.

Примеры логики

Персонаж ходит по пути **walk1**, а при приближении игрока на дистанцию 5 метров переключается на путь **walk2** (но только при условии, что он видит игрока):

```
[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 все равно будут обработаны. Если все условия в фигурных скобках не выполняются, секция активирована не будет.

Например:

on_actor_dist_le = 5 | {условие} walker2 %эффекты%

Условия

- **+infoportion**
  Требуется присутствие infoportion у actor-а.
- **-infoportion**

Требуется отсутствие infoportion у actor-а.

- **=func**
  Требуется, чтобы func вернула true.
- **!func**
  Требуется, чтобы func вернулся false.

Эффекты

- **+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 условий или эффектов разделяются пробелами:

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** равносилен 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 нужно, если какое-либо условие, выполнившись один раз, больше не нуждалось 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.

> **Заметка**
>
> Обратно из секции **nil** под скрипты объект вернуть уже невозможно! Учитывайте это, используя её.

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 её пошагово. Вначале сталкер работает по схеме **walker**. 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 idle-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, то 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 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).

Схемы логики space_restrictor

Общее замечание

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

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. Впрочем, **nil** можно не задавать.

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

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]
```

Секция sr_light

Зона, в которой фонарики у неигровых персонажей будут включены независимо от времени суток.

Работает следующим образом:

```
[logic]
active = sr_light

[sr_light]
light_on = true
```

Также работает вместе с кондлистом:

```
[logic]
active = sr_light

[sr_light]
light_on = false
on_info = {+info1} section %+info2%
```

Схема sr_particle

Данная схема отыгрывает частицы (как статичные, так и движущиеся) в указанном месте и в указанное время.

[sr_particle]

- **name = ...**
  Имя системы частиц.
- **path = ...**
  Имя пути камеры.
- **mode = ...**
  Режим проигрывания частиц:
  - **1** — с путем камеры
  - **2** — с обычным патрульным путем
- **looped = true**

Флаг зацикленности (**true**/**false**).

Пример использования — система частиц с обычным патрульным путем:

```
[sr_particle]
name = explosions\campfire_03
path = part_points
mode = 2
looped = true
```

В точках пути можно задавать флаги **s = имя_звуковой_темы** и **d = число** время задержки перед проигрыванием (задаются 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 одновременно с частицами. Звук не зацикливается и играет только один раз. Результат — частицы отыгрываются во всех waypoint-ах одновременно (или с задержкой, см. выше).

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

В схеме поддерживается кондлист. Если рестриктор переходит в другую секцию, то автоматически перестают отыгрываться частицы и звуки для них. Этот рестриктор является объектом, отслеживающим частицы и нет никакой необходимости, чтобы игрок в него заходил.

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%
```

- **type = ...**
  Тип счетчика — инкриментирующий (inc) или декриментирующий(dec). Если поле не задано, то счетчик будет инкриментирующим.
- **start_value** - начальное значение счетчика в миллисекундах *реального* времени. Для декриментирующих счетчиков задавать обязательно. Для инкриментирующих, 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%**

Схема sr_psy_antenna

Зоны с такой секцией позволяют управлять эффектами от пси-воздействия (на Янтаре и Радаре). Также 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 процентах от базового значения интенсивности излучения.
- **hit_ intensity = ...**
  Увеличение/уменьшение в процентах 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
```

Схема sr_cutscene

В нашем движке можно воспользоваться камерой для создания катсцен.

```
[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
look = agru_nv_camera_look
cam_effector = scenario_cam\agroprom_underground\camera1_0_904
on_signal = cameff_end | sr_cutscene@cam2
global_cameffect = true
```

```
[sr_cutscene@cam2]
…
```

- **point = …**
  Точка walk, где игрок будет находится после камеры.
- **look = …**
  Точка look, куда будет направлена камера игрока после камеры.
- **cam_effector = …**
  Файл камеры и путь к нему.
- **on_signal = …**
  Сигнал окончания камеры и переход в следующую секцию.
- **global_cameffect = …**
  Флажок отыгрывания камеры от глобальных координат уровня, а не от камеры игрока (по умолчанию false).

Ещё один пример использования:

```
[logic]
active = sr_idle

[sr_idle]
on_info = {!black_screen} sr_cutscene@cam

[sr_cutscene@cam]
point = mil_nv_camera_walk
look = mil_nv_camera_look
cam_effector = scenario_cam\mp_military_weather\mp_military_weather
on_signal = cameff_end | sr_cutscene@cam %=exit_game%
on_timer = 862000 | %=run_postprocess(agr_u_fade)%
```

Заметка

Камера от рестриктора будет работать лишь в случае, если:
• на момент ее запуска игрок находится внутри данного рестриктора
• рестриктор имеет тип **NONE default restrictor** в LE

Набор дополнительных настроек логики у разных объектов

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

Секция ph_door (схема работы двери)

Заметка

Для двухстворчатых ворот задается все аналогично.

- **locked = true/false**
  Заперта ли дверь. По умолчанию false.
- **closed = true/false**
  Закрыта ли дверь. По умолчанию true.
- **tip_open = ...**
  Подсказка, которая появляется около прицела при наведении на дверь, если дверь закрыта. Если **locked == false**, то **tip_door_open**, иначе **tip_door_locked**.
- **tip_close = ...**
  Подсказка, которая появляется около прицела при наведении на дверь, если дверь открыта. Если **locked == false**, то **tip_door_close**, иначе пустое значение.
- **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.

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
```

Секция ph_button (схема работы кнопки)

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 отыгрывается при нажатии на кнопку.
- **anim_blend = ...**
  Плавная, сглаженная анимация (true/false).
- **tooltip = ...**
  Текстовая подсказка (выводится при наведении на кнопку).

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, его 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%
```

Секция ph_gate (схема работы ворот)

То же самое, что и **ph_door**, но для ворот из двух половин.

- **state = ...**
  Состояние, в котором дверь находится при инициализации (по умолчанию **none**):

- **open** — в открытом;
- **closed** — в закрытом;
- **none** — в текущем (стандартном 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.

- **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**.

Пример использования:

```
[ph_gate@locked]    ;блокировка в открытом состоянии, неразбиваемые.
state = opened
locking = soft
left_limit = 130
rigt_limit = 60
breakable = false

[ph_gate@opened]
state = opened
locking = stick

[ph_gate@closed]
state = closeded
```

Секция ph_code (кодовые замки)

При введении указанного кода выдает инфопоршн.

```
[logic]
active = ph_code@lock

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

Секция ph_force (толкнуть физический объект)

Схема позволяет пнуть предмет в указанную сторону. Прописывается в custom_data предмета.

- **force = ...**
  Сила, прикладываемая к объекту. Измеряется в убитых енотах.
- **time = ...**
  Время приложения 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.

Секция ph_heavy (запретить кидать объект)

Прописывается в custom_data физических объектах, которые запрещено для швырять бюрерам и полтергейстам. Например, если они должны лежать на конкретном месте (типа сюжетных документов) или слишком громоздки по габаритам, чтобы их можно было красиво кидать.

Секция ph_oscillate (раскачивание физики)

Схема предназначена для плавного раскачивания физики (лампы, висящие зомби и т.д.).

- **joint = ...**
  Имя кости, к которой будет применена сила.
- **force = ...**
  Собственно сила (в ньютонах).
- **period = ...**
  Время приложения силы.

Пример использования:

```
[logic]
active = ph_oscillate

[ph_oscillate]
joint = provod
force = 1.5
period = 500
```

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

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%**
- **sound_type**
  Тип звука:
  - **WPN_hit** — звук попадания пули
  - **WPN_reload** — звук перезарядки оружия
  - **WPN_empty** — звук попытки выстрелить из незаряженного оружия
  - **WPN_shoot** — звук выстрела
  - **MST_die** — звук смерти
  - **MST_damage** — звук получения урона
  - **MST_step** — звук шагов

Переход состоится, если обьект услышит звук **sound_type** от обьекта **story_id** при дистанции **<= distance** и силе звука **>= sound_power**.

Также поддерживается запись:

- **on_sound1 = ...**
- **on_sound2 = ...**

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

Пример использования:

```
[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 всё, 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    ; Услышит примерно с 3х метров
on_sound7 = zat_cop_id|WPN_reload|10|0.3| mob_home@fight    ; Услышит примерно с 3х метров
```

Управление некоторыми особенностями

meet_manager (настройка реакции NPC)

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
abuse = true/false
trade_enable = true/false    ;По умолчанию true. Можно ли торговать с NPC
allow_break = true/false    ; По умолчанию false. Можно ли выйти из диалога
quest_npc = 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, когда 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 SID-а персонажа, на которого нужно будет смотреть.
- **use**, **use_wpn**
  Настройки юзабельности персонажа. Возможны три варианта:
  - **true** — с NPC можно говорить и торговать;
  - **false** — с NPC нельзя говорить и торговать;

    - **self** — NPC сам юзнет игрока, как только сможет дотянуться.
- **zone**
  Содержит набор имен рестрикторов, а также анимаций и озвучки, которую NPC будет отыгрывать, если игрок будет замечен в рестрикторе.
- **meet_dialog**
  Стартовый диалог NPC.
- **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**
  Может ли NPC обижаться на actor-а. 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 в нас стрелять.

```
[meet]
meet_state = 50| {+info} threat_fire %=killactor%, walk@ {+info} talk_abuse, wait | 10 | walk %+info%;
wait | 2 | threat;state  ;WTF???
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
```

- 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 xr_effects, которая обижает NPC 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
```

- 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.

По умолчанию встреча настроена со следующими параметрами:

```
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
```

Заметка

Если нужно, чтобы сталкер не разговаривал с игроком в данной секции, необходимо прописать ему:
• **meet = no_meet**

Категории:

- Pages with syntax highlighting errors
- Страницы с неработающими файловыми ссылками
- Чистое небо
- Зов Припяти
