

## **3. Настройки логики**

### 3.1. Система флагов path\_walk, path\_look

3.1.1. Более подробное описание путей.

### 3.2. Схемы поведения сталкеров

3.2.1. Walker

3.2.2. Remark

3.2.3. Sleeper

3.2.4. Камп

3.2.5. Camper

3.2.5.1. Sniper

3.2.6. Follower (Отключен)

3.2.7. Zoneguard

3.2.8. Wounded

3.2.9. Rest

### **3.2.10. Схема heli\_hunter**

3.2.11. Patrol

### 3.3. Секции

3.3.1. Combat

3.3.2. Death

3.3.3. Hit

3.3.4. Actors\_dialog

3.3.5. Use

3.3.6. Combat\_ignore

3.3.7. Секция dont\_spawn\_character\_supplies

3.3.8. Секция no\_smart

3.3.9. Treshhold

3.3.10. Danger

3.3.11. Истории у костров

### 3.4. Оверрайды

### 3.5. Схемы поведения для монстров

3.5.1. Mob\_walker

3.5.2. Mob\_eluder

3.5.3. Mob\_remark

3.5.4. Mob\_combat

3.5.5. Mob\_death

3.5.6. Mob\_jump

3.5.7. Mob\_camp

3.5.8. Mob\_home

3.5.9. Mob\_fake\_death

### 3.6. Оверрайды для монстров

### 3.7. Секция спавнер

3.7.1. Спавн дневных и ночных монстров

### 3.8. Скрипт Logic

3.8.1. Синтаксис logic-a

3.8.2. Примеры достаточно сложной логики

### 3.9. Схемы space\_restrictor

- [3.9.1. Sr\\_idle](#)
- [3.9.2. Sr\\_no\\_weapon](#)
- [3.9.3. Sr\\_sound](#)
- [3.9.4. Sr\\_tip](#)
- [3.9.5. Sr\\_light](#)
- [3.9.6. Sr\\_territory](#)
- [3.9.7. Sr\\_mapspot](#)
- [3.9.8. Sr\\_particle](#)
- [3.9.9. Sr\\_sound\\_act](#)
- [3.9.10. Sr\\_timer](#)
- [3.9.11. Sr\\_psy\\_antenna](#)
- [3.9.12. Sr\\_teleport](#)
- [3.9.13. Sr\\_sleep и настройка снов](#)
- [3.9.14. Sr\\_cutscene](#)
- [3.10. Дополнительные настройки логики у разных объектов](#)
  - [3.10.1. Ph\\_door](#)
  - [3.10.2. Ph\\_button](#)
  - [3.10.3. Работа прожектора](#)
  - [3.10.4. Ph\\_code](#)
  - [3.10.5. Ph\\_gate](#)
  - [3.10.6. Ph\\_sound](#)
  - [3.10.7. Ph\\_force](#)
  - [3.10.8. Ph\\_on\\_death](#)
  - [3.10.9. Ph\\_car](#)
  - [3.10.10. Ph\\_heavy](#)
  - [3.10.11. Ph\\_oscillate](#)
- [3.11. Смарттерейны и гулаги](#)
  - [3.11.1. Смарттерейны](#)
    - [3.11.1.1 Стандартный набор смарттерейнов](#)
  - [3.11.2. Гулаги.](#)
  - [3.11.3. Новые особенности смарттерейнов.](#)
    - [3.11.3.1. Более доступное описание новых смарттерейнов](#)
- [3.12. Логика вертолета](#)
  - [3.12.1. Схема heli\\_move](#)
  - [3.12.2. Универсальная боевая схема \(Отключен\)](#)
- [3.13. Meet\\_manager](#)
- [3.14. Отметки на минимале](#)
- [3.15. Передача параметров в функции.](#)
- [3.16. Настройка звуковых групп.](#)

### **3.1. Система флагов (path\_walk, path\_look)**

В точках путей можно задавать флаги, изменяющие поведение персонажа. Флаги задаются прямо в имени waypoint-а, 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`-а `state manager`-ом. Соответственно, если `t=0`, то сигнал будет установлен после отыгрыва `init` анимации. Это используется, например, с анимацией `press`, которая состоит из двух частей: 1 - нажимаем на кнопку, 2 - опускаем руку.

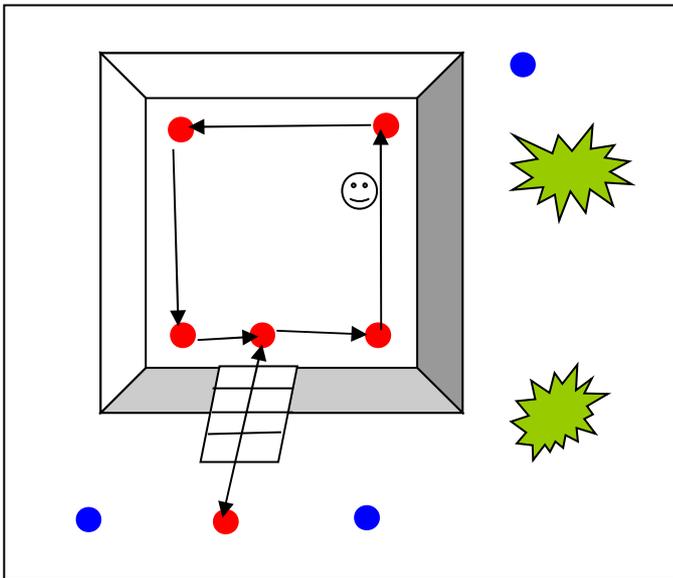
В пути `path_look` можно сделать: `wp00|a=press|t=0|sigtm=preserved`

А затем переключить схему: `on_signal = preserved | другая_схема`

### 3.1.1. 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 = ...`

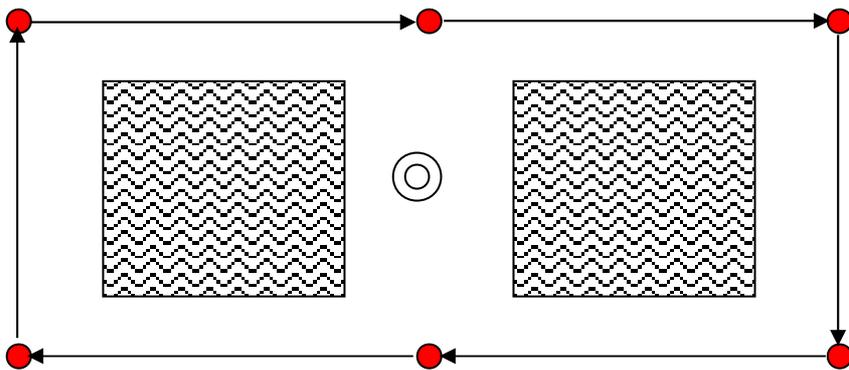
(не обязательно) имя пути, описанного в п. 2. Если персонаж должен только **ходить** по маршруту, `path_look` можно не задавать.

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:



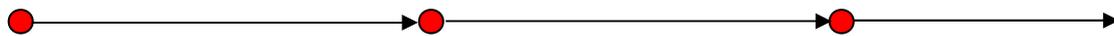
**wp00** персонаж идет      **wp01|a=raid** бежит в danger-е

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:



**wp00|s=state|sc** говорит      **wp01|s=state|sc** говорит      **wp02** молчит

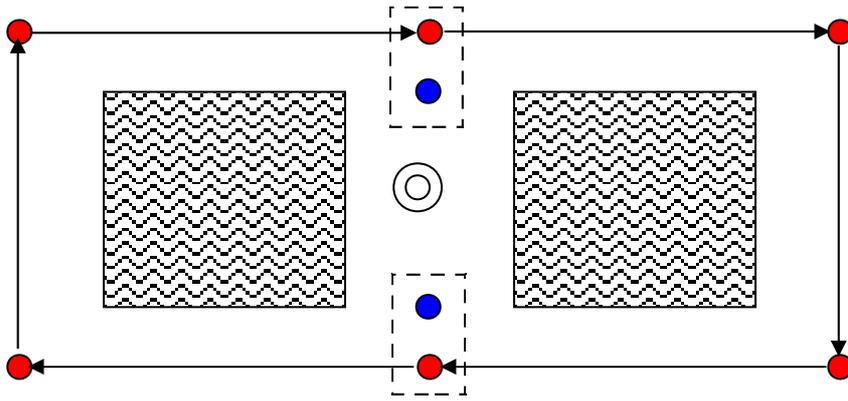
В примере 3 используется только поле s, чтобы задать тему разговора, и флажок sc, чтобы показать, что звук проигрывается не разово, а периодически.

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:

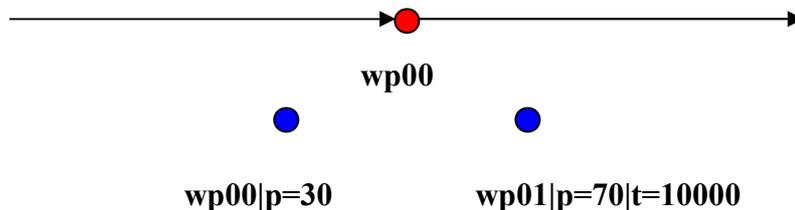
$p = 100$  – вероятность, с которой персонаж посмотрит именно в эту точку. Значения  $p$  всех подходящих точек суммируются, т.е. если у одной точки  $p = 100$ , а у другой 300, то персонаж посмотрит в первую с вероятностью 25%! (т.е. 100 из 400).

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 **squish**, либо анимацию **hide**, если он в состоянии **squish**.

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»!

## 3.2. Схемы поведения сталкеров.

Есть определенный набор схем, которые описывают поведение персонажа. Они прописываются у него в custom\_data или, в случае гулага, в соответствующих файлах, описывающих работы данного гулага. Ниже приведен перечень этих схем.

В файле \gamedata\scripts\modules.script указаны все загружаемые схемы.

### 3.2.1. Схема walker

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

[walker]

path\_walk = <имя пути>- основной путь, по которому ходит NPC

\*path\_look = <имя пути>- путь, куда смотрит NPC

\*team - команда для синхронизации

В точках path\_walk, которым соответствуют точки пути path\_look (стоят одинаковые флажки) персонаж останавливается и смотрит в определенную точку, при этом отыгрывая (или не отыгрывая) определенную анимацию.

\* def\_state\_moving1 = состояние, 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.

Файл: \gamedata\scripts\xr\_walker.script

### 3.2.2. Схема remark

Схема используется для синхронизации\связки других схем.

[remark]

\*snd\_anim\_sync = true либо false. По умолчанию false. Указывает на то необходимо ли синхронизировать звук с анимацией либо нет

\*snd = звук ремарка, по умолчанию nil

\*anim = анимация ремарка, по умолчанию wait

\*target = Куда смотрит сталкер. Есть следующие варианты

story\_id – числовое значение

actor – без комментариев

nil – позиция вычисленная АИ автоматически

<имя работы>, <имя гулага> смотреть на сталкера который находится на определенной работе под гулагом (второй параметр необязателен. В этом случае берется гулаг сталкера, для которого задана данная секция ремарка).

Пример:

```
target = logic@cit_killers_base_guard, cit_killers
```

<path\_name>, <point\_number> - можно указывать смотреть в вершину патрульного пути (<имя пути>, <имя точки>).

Внимание, теперь если значение не задано, то оно равно nil а не actor, как было раньше. То есть если вы хотите чтобы персонаж в ремарке смотрел на актера - необходимо явно прописывать это. Если задано значение nil, то персонаж развернется в позицию, которую посчитает АИ.

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
```

### 3.2.3. 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).

Файл: \gamedata\scripts\xr\_sleeper.script

### 3.2.4. 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)
```

Файл: \gamedata\scripts\xr\_kamp.script

**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 точке, 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.

### 3.2.5. 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 направлении 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`

`*radius = number` – расстояние в метрах, если расстояние между кемпером 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. Ставьте кемперов с вышеуказанным флажком. Они идут по своим патрульным путям и выносят врагов.

`*def_state_moving` = состояние из стейт менеджера

Состояние, в котором мы движемся на ближайшую точку пути при враге

`*def_state_moving_fire` = состояние из стейт менеджера (`sneak_fire`)

Состояние, в котором мы отстреливаемся от врага, во время движения на ближайшую точку пути.

`*def_state_campering` = состояние из стейт менеджера (`hide`)

Состояние, в котором мы ожидаем врага, находясь на пути

`*def_state_campering_fire` = состояние из стейт менеджера (`hide_fire`)

Состояние, в котором мы отстреливаемся от врага, находясь на пути

`*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` = тип.

Задаем тип стрельбы. Возможные значения - `always|none|terminal`

`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.

**NB!** 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 `walkеров` в таких случаях, а кемперов стоит ставить для атак по путям и как снайперов.

### 3.2.5.1. 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 кастом дате кемпера прописать:

```
sniper = true
```

Файл: \gamedata\scripts\xr\_camper.script

### 3.2.6. Схема follower

В custom\_data прописан как follower

NPC идет за NPC лидером. Если до лидера расстояние менее 5 метров, то он идет, если от 5 до 20 – бежит в режиме run, если свыше 20 – догоняет в режиме sprint. Пути не задаются.

```
[follower]
```

```
leader = story id лидера из game.ltx (число!)
```

```
*formation_line = true (постарается идти сбоку от лидера, в противном случае будет идти сзади
```

```
*distance = расстояние в метрах, на котором будет идти от лидера attendant. По умолчанию – 1,5 метра, если идет цепью, то 5 метров.
```

```
*state_if_leader_in_meet. Это есть строка с именем состояния из state_manager, которое будет назначено follower-ам, если командир пребывает в состоянии meet.
```

```
*anim_walk = state (состояние, в котором фолловер идет за лидером)
```

```
*anim_run = state (состояние, в котором фолловер бежит за лидером)
```

```
*anim_sprint = state (состояние, в котором фолловер спринтует за лидером)
```

```
Файл: \gamedata\scripts\xr_attendant.script
```

Если все это происходит под гулагом, то вместо story\_id лидера, мы прописываем его секцию логики в файле скрипта. Пример:

```
t = { section = "logic@bar_arena_follower_2",
      idle = 0,
      prior = 7, state = {0}, squad = squad, group = groups[0],
      in_rest = "", out_rest = "",
      dependent = "logic@bar_arena_leader",
      predicate = function(obj)
                    return obj:character_community() == "dolg"
                end
    }
```

### 3.2.7. Схема zoneguard

NPC есть две зоны (может быть одна). Он ходит по путям, но когда игрок заходит в зону, отрывает от дел, подбегает к игроку, наставляет на игрока оружие (может кричать, может говорить), если игрок заходит во вторую зону – атакует игрока

```
[zoneguard]
```

```
path_walk = путь перемещения
```

```
*path_look = путь обзора
```

```
team = имя команды синхронизированных zoneguard-ов (из всей команды только 1 будет реагировать на игрока)
```

```
*zone_guard = имя зоны, в пределах которой игрок будет атакован
```

```
zone_warn = имя зоны, в пределах которой начинать разговор с игроком
```

\*walker\_team = team для схемы перемещения его в состоянии walker (если не задан, используется значение из поля team)  
 \*no\_move = если true, персонаж окликнет игрока с места и не будет подбегать к нему  
 \*snd\_greet = имя звуковой схемы, из которой будет проигран звук при обнаружении персонажа  
 \*ignore\_friends = true, будет игнорировать дружественных ему персонажей.  
 \*ignore\_cond = {+info -info =func !func} условия, при которых NPC игнорирует игрока  
 \*no\_danger = если true, то не отыгрывает угрожающую анимацию, нейтралам.  
 \*anim = какую отыгрывает анимацию, если игрок ему не враждебен.  
 \*snd\_anim\_sync = если true, то npc будет синхронизировать звук с анимацией  
 Файл: \gamedata\scripts\xr\_zoneguard.script

### 3.2.8. Схема wounded (раненый)

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

```
[walker]
wounded = wounded
```

```
[wounded]
hp_state      = HP|condstate@condsound|HP|condstate@condsound
hp_state_see  = HP|condstate@condsound|HP|condstate@condsound
psy_state     = PSY|condstate@condsound|PSY|condstate@condsound
hp_victim     = HP|condvictim|HP|condvictim
hp_cover      = HP|condbool|HP|condbool
hp_fight      = HP|condbool|HP|condbool
*syndata      = state@sound|state@sound
*help_dialog  = story_id
*help_start_dialog = story_id
```

Где:

**Condstate** – кондлист, возвращающий состояние персонажа, либо true. Если он возвращает true – npc обидится на игрока

**Condsound** – кондлист, возвращающий саунд тему.

**HP** – пороговое значение здоровья персонажа

**PSY** – пороговые значения пси здоровья персонажа

**Condvictim** – кондлист, возвращающий направление куда смотреть. Возможные значения: nil, actor, number. В случае числа – будет смотреть на персонажа с указанными стори айди.

**Condbool** – кондлист, возвращающий true либо false.

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

hp\_cover – идти в укрытие или нет, в зависимости от ХП

hp\_fight – разрешено воевать или нет, в зависимости от ХП

syndata – синхронпары для красоты.

help\_dialog – story\_id диалога вместо дефолтного actor\_help\_wounded. Если вам по сюжету необходимо заменить диалог другим, то вы в этом поле прописываете id 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.

**Пример.** В качестве примера взята дефолтная настройка.

```
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.

Файл: \gamedata\scripts\xr\_wounded.script

### 3.2.9. Схема rest

Чувак гуляет, хакает, спит.

Пока нормально не работает.

Файл: \gamedata\scripts\xr\_rest.script

### 3.2.10. Схема heli\_hunter

Хелихантер может стрелять либо не стрелять по вертолету в зависимости от условий. Делается это так:

```
[camper@bar_freedom_attack_sniper_1]
path_walk = camper_1_walk
path_look = camper_1_look
on_info = {+bar_freedom_attack_ecolog} camper1@bar_freedom_attack_sniper_1
%=bar_freedom_angry_actor%
meet_talk_enabled = true
meet_dialog = bar_svoboda_dialog
heli_hunter = {-bar_ecolog_crush_heli_down} true, false
```

Если раньше оверрайд хелихантера понимал только значения true либо false, то сейчас он понимает кондлист, который если возвращает true - то стрельба по вертолету в данной схеме разрешена.

### 3.2.11. Patrol

Итак, есть предварительная система патруля. Представляет собой вариацию kamp только в состоянии ходьбы. Для ее работы прописываем в кустовой дате следующее:

```
[patrol]
path_walk = path_walk
path_look = path_look
*formation = back
*commander = true (типа назначат командиром, желательно, чтобы такой красивый он был один)
*move_type = задает изначальный режим перемещения, по умолчанию patrol. Вообще, значение этого поля есть название ходячей анимации из state_mgr_lib
```

formation - 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 командор помирает, то автоматически будет выбран другой. Командиром становится тот, кто первый попал под схему. Способы построения задаются в вейпоинтах следующим образом:

get=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.

Что еще не сделано или глючит:

- нет возможности автоматически перестроить команду (нужно от Шурика то, что записано в todo листе)

- все идут молча (когда будет менеджер баек, то сделаем)

- командор пока не отдает команд (нет озвучки)

- не рекомендуется включать спринт (глючит)

### 3.3. Секции.

#### 3.3.1. Секция combat

Показывает, что происходит, когда NPC срывается в бой.

```
on_combat = combat
```

```
[combat]
```

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

Для задания различных типов скриптовых боёв для различных ситуаций используется параметр combat\_type.

В следующем примере сталкер сражается:

\* по-кемперски, если враг=актёр и он дальше X метров

\* по-монолитовски, если любой враг дальше Y метров

\* иначе - движковый бой

```
[logic]
```

```
active = walker
```

```
on_combat = combat
```

```
[walker]
```

```
path_walk = ...
```

```
[combat]
```

```
combat_type = {=fighting_actor =fighting_ge_X_meters} camper, {=fighting_ge_Y_meters} monolith
```

Пример такой функции: нам надо чтобы на расстоянии свыше 20 метров npc переходил бы в кемперский комбат.

```
function fighting_dist_ge_20(actor, npc)
```

```
return db.storage[npc:id()].enemy:position():distance_to ( npc:position() ) >= 400
```

end

400 – это  $20^2$ . Примечание – мы пишем квадрат нужного нам расстояния, для экономии системных ресурсов.

Ещё один пример. Сталкер ходит под симуляцией, но у него бой не движковый, а всегда зомбированный:

```
[logic]
active = nil
on_combat = combat

[combat]
combat_type = zombied
```

Если в разных секциях для персонажа требуются разные типы боя или разные условия, то можно воспользоваться оверрайдом `combat_type`.

Помните: оверрайд всегда будет перекрывать настройку в секции `combat`. Т.е., если у вас логика на 5 секций и в четырёх нужен кемперский комбат, а в пятой монолитовский, то можно задать так:

```
[logic]
active = walker1
on_combat = combat

[walker1]
...
[walker2]
...
[walker3]
...
[walker4]
...
[walker5]
...
combat_type = monolith

[combat]
combat_type = camper
(scheme - задает тип боя (monolith, camper, zombied), иначе - универсальный бой)
```

`disable_combat_handler` – функция отключающая секцию `combat`.

Файл: `\gamedata\scripts\xr_combat.script`

### 3.3.2 Секция `death`

Схема показывает, что происходит при смерти NPC.

```
on_death = death
```

```
[death]
on_info = %+info -info =func%
Файл: \gamedata\scripts\xr_death.script
```

### 3.3.3. Секция hit

Схема показывает, что происходит при, нанесении повреждения NPC. on\_hit НЕ СРАБАТЫВАЕТ на звук выстрела, только на попадание по сталкеру! Это сделано, потому что выстрел в воздух в общем случае не должен восприниматься как агрессия (игрок отстреливает, скажем, собак, а на него срывается охрана).

```
on_hit = hit
```

```
[hit]
```

```
on_info = %+info -info =func%
```

```
Файл: \gamedata\scripts\xr_hit.script
```

### 3.3.4. Секция actor\_dialogs

Показывает, какие диалоги будут доступны или недоступны игроку при разговоре с этим NPC. Пишется практически в любой схеме.

```
actor_dialogs = actor_dialogs
```

```
[actor_dialogs]
```

```
id = доступные диалоги через запятую.
```

```
disable = запрещенные диалоги, тоже через запятую.
```

```
Файл: \gamedata\scripts\xr_meet.script
```

### 3.3.5. Секция use

Схема показывает, что произойдет, если игрок юзнет NPC.

```
on_use = use
```

```
[use]
```

```
on_info = %+info -info =func%
```

```
Файл: \gamedata\scripts\xr_use.script
```

### 3.3.6. Секция combat\_ignore

Если NPC в этой схеме то он, не переходит в боевой режим. В любой другой схеме:

```
[walker]
```

```
combat_ignore_cond = {+info -info =func !func} – условия для игнорирования боя (если написать always, то в данной схеме игрок будет игнорировать бой всегда, пока не перейдет в схему, где бой не игнорируется).
```

В схеме нет дополнительных полей

```
[walker]
```

```
combat_ignore = combat_ignore
```

```
[combat_ignore]
```

Функции, используемые для работы с кондлистом комбат игнора:

fighting\_dist\_ge\_20 -- текущий враг на расстоянии больше или равном 20м  
fighting\_dist\_ge(расстояние в метрах) – универсальная функция для combat\_ignore, проверка расстояния для игрока

fighting\_actor -- текущий враг актёр?  
check\_fighting -- проверка (по story\_id) того, что нашим врагом есть хотя бы кто-то один из списка

Файл: \gamedata\scripts\xr\_combat\_ignore.script

### 3.3.7. Секция dont\_spawn\_character\_supplies

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

[dont\_spawn\_character\_supplies]

### 3.3.8. Секция no\_smart

Если прописана эта секция, то прс не берется под смарттеррейн даже если он походит по всем параметрам.

[no\_smart]

### 3.3.9. Секция treshhold

Есть возможность изменять у сталкеров параметры, по которым они атакуют монстров. Этим параметра два:

max\_ignore\_monster\_distance (в данный момент дефолт 15 метров). Сталкер будет всегда атаковать монстров, которые находятся внутри данного радиуса.

ignore\_monstre\_threshold (в данный момент дефолт 0). Параметр от 0 до 1. Если функция оценки монстра ниже, чем этот параметр, и монстр находится за пределами вышеуказанного радиуса - он будет атакован. В данный момент все настроено так, что сталкеры вообще не атакуют монстров находящихся дальше чем 15 метров от них.

В секции логики либо в текущей схеме указываете:

```
threshold = threshold@tratata
```

```
[threshold@tratata]
```

```
max_ignore_distance = <number>
```

```
ignore_monster = <number>
```

Второй параметр следует менять **ОЧЕНЬ** осторожно.

### 3.3.10. 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:

```
danger_inertion_time_grenade =
danger_inertion_time_corpse =
danger_inertion_time_hit =
danger_inertion_time_sound =
```

Дефолтовые настройки:

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

**NB!!** Также эти настройки теперь распространяются и на схему кемпера. То есть в настройках кемпера перестало работать поле `danger_radius`. Теперь данные берутся из секции денжера согласно общих правил.

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.

В данный момент установлены следующие дефолты:

```
ignore_distance = 50
ignore_distance_grenade = 15
ignore_distance_corpse = 10
ignore_distance_hit = 50
ignore_distance_sound = 50
```

**NB:** если надо, чтобы в разных случаях сталкер игнорировал разные типы данжеров, создается несколько секций данжера `danger_condition@1`, `danger_condition@2` и так далее.

\* `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.

### 3.3.11. Байки из склепа (Истории у костра)

Из нового: теперь лагерь автоматически рассказывать истории не будут. Для этого вы должны того или иного сталкера "научить" истории.

Делается это так: в кастом дате пишется секция:

```
[game_info]
stories = "story_01, legend_01"
```

В кавычках список историй и легенд через запятую. Пока что существуют следующие истории и легенды:

story\_01 - Граница зоны и граната за 1 действие.

story\_02 - Про трамплин и про камешки

story\_03 - Про то как группа Вильнова вернулась

story\_04 - Про то как Костя Федорин наткнулся на артефакт и пропал на радаре.

story\_05 - Про то как духманам с контролером стражаться.

story\_06 - Про дверцу, водку и избушку.

legend\_01 - Про эксперимент в Зоне, который производят инопланетяне.

legend\_02 - Об особо засекреченных лабораториях в зоне.

legend\_03 - Легенда о проводнике

legend\_04 - Легенда о темном сталкере

legend\_05 - Легенда о том что глубоко в Зоне спать нельзя.

О том какие истории и легенды в каком лагере на каком уровне можно и нельзя юзать узнавать о Профа.

### 3.3.12. 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]

## 3.4. Оверрайды:

Настройки, которые меняют поведение общих схем, в зависимости от активной в данный момент обычной схемы (все они необязательны)

\*meet\_enabled = true (запускает схему встречи)

\*meet\_talk\_enabled = true (в действующую схему поведения добавляет возможность диалога)

\*meet\_dialog = <название диалога>, который будет запущен при юзе.

\*meet\_state = <название состояния> он определяет, в каком состоянии будет находиться

персонаж, если открылось диалоговое окно общения и торговли

\*wounded\_enabled = true (включает NPC возможность использовать схему раненого)

\*combat\_ignore\_cond = см. выше

\*combat\_ignore\_keep\_when\_attacked = true (игрок продолжает игнорировать бой, даже если в него стреляют – ТОЛЬКО В СЛУЧАЕ СТРЕЛЬБЫ ИГРОКА!!!!)

\*combat\_type = {условие} scheme - тип боя которым будет пользоваться прс из данной схемы

\*on\_combat = см. выше

\*companion\_enabled = true (свободноходящие сталкеры могут наниматься как компаньоны (в будущем они будут брать за это деньги)).

\*invulnerable = true (делает персонажа неуязвимым).

## 3.5. Схемы для монстров

### 3.5.1. 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) с - идти дальше в присяге r - дальше бежать 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=anim\_set - анимация (stand\_idle, sit\_idle, lie\_idle, eat, sleep, rest, attack, look\_around, 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
```

Задает 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
```

### 3.5.2. Схема mob\_eluder

Монстр перемещается по точкам патрульного пути (не учитывая связи между точками), держась на расстоянии от игрока, при этом придерживаясь своего пути, выходя из под схемы при слишком близком приближении к игроку, и возвращаясь обратно, когда расстояние увеличится.

path = ... работает как обычно path\_walk. Набор точек патрульного пути.

\*Time\_capture = .... (время в секундах) время, которое монстр находится под этой схемой. Default – 10.

\*Time\_release = .... (время в секундах) время, которое монстр находится под универсальной схемой. Default – 10.

\*Min\_dist = .... (расстояние в метрах, если расстояние до врага меньше этого, то он переходит под универсальную схему). Default – 5.

\*Max\_dist = .... (расстояние в метрах, если расстояние до врага больше этого, то он переходит под eluder). Default - 10

Замечание – работает нестабильно.

Файл: \gamedata\scripts\mob\_eluder.script

### 3.5.3. 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.  
Файл \gamedata\scripts\mob\_remark.script  
На этой схеме сделан торговец.

### 3.5.4. Схема mob\_combat, mob\_death

Работают точно также как и у сталкеров соответствующие схемы.

Файлы: \gamedata\scripts\mob\_combat.script, \gamedata\scripts\mob\_death.script

### 3.5.6 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)

### 3.5.7. Mob\_camp

Механика:

1. Сидит на позиции, смотрит в точку
2. Можно задать несколько позиций и время смены позиции.
3. Перемещается между позициями бегом
4. При виде врага переходит под универсальную схему (комбат/паника и т.д)
5. Задаются минимальная и максимальная дистанции от врага до текущей camp-позиции

6. Если враг уходит далеко - монстр возвращается на позицию

Использование:

```
[logic]
```

```
active = mob_camp
```

```
[mob_camp]
```

```
path_look = way_look
```

```
path_home = way_home
```

```
time_change_point = 30000
```

```
home_min_radius = 20
```

```
home_max_radius = 50
```

skip\_transfer\_enemy – если прописать в кастом дату, то монстр не будет принимать врага от друших монстров, если его увидит (для этого нужно всех монстров в разные group разнести)

Описание параметров:

\*path\_home - путь, состоящий из точек, в которых будет находиться монстр

path\_look - путь, состоящий из точек, в которые будет смотреть монстр

\*time\_change\_point - время изменения текущей camp-точки (по-умолчанию 10000), мс

\* home\_min\_radius - минимальный радиус от врага до camp-точки (по-умолчанию 30), м

\* home\_max\_radius - максимальный радиус от врага до camp-точки (по-умолчанию 40), м

Особенности:

Минимальный и максимальный радиус необходимы для игнорирования врага, если он убежал далеко и для возврата на текущую позицию. Учитывается дистанция от врага до текущей позиции. Если дистанция меньше home\_min\_radius - атакуем врага, пока враг не исчезнет или дистанция не будет больше home\_max\_radius.

Две дистанции необходимы для того, чтобы избежать ситуации, когда игрок стоит на границе радиуса действия и входит/выходит в зону и монстр бежит то в свою camp-позицию, то на врага.

Выбор текущей позиции производится случайным образом

Индексы точек пути для path\_home и path\_look должны совпадать (т.е. монстр сидит во второй точке path\_home и смотрит во вторую точку path\_look)

Единственным необходимым параметром является path\_look

Если не установлен path\_home, в качестве кемперской точки учитывается позиция и нода объекта на спауне.

Для того чтобы монстр смотрел в разные точки на кемпер-позиции, path\_look может состоять из нескольких точек.

Обязательные требования:

home\_min\_radius < home\_max\_radius

Количество точек путей path\_look и path\_home должно быть равным

P.S. Mob\_Camp можно использовать как альтернативу к монстрам под рестрикторами

### 3.5.8. Mob\_home

Схема является ещё одним решением по замене рестрикторов. Рекомендую все следующие гулаги монстров делать на mob\_home, а старые гулаги постепенно переводить на mob\_home. У кого рестрикторы работают хорошо и красиво, их можно не трогать.

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 айidle 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.

### 3.5.9. Mob\_fake\_death

Появилась схема mob\_fake\_death для зомби. Необходимо для сценок, когда игрок идёт, а вокруг него начинают подниматься зомби...

Использование:

```
[logic]
```

```
active = mob_fake_death
```

```
[mob_fake_death]
```

```
on_actor_dist_le = 5 | nil
```

При входе в схему зомби падает, при выходе из схемы встает.

### 3.6. Оверрайды для монстров:

actor\_friendly = если true, то монстр не атакует актера, до первой атаки на него

npc\_friendly = если true, то монстр не атакует сталкеров и монстров, до первой атаки на него

friendly = если true, то монстр не атакует никого до первой атаки на него

braindead = если true, то монстр игнорирует любые атаки.

Секции для монстров

```
[mob_death], [mob_hit]
```

### 3.7. 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

### 3.7.1. Спавн монстров дневных и ночных.

```
[spawner]
cond = {=is_day} – спавнить монстра только днем (если надо ночью, то пишем {!is_day})
check_distance = true – проверка на наличие персонажа рядом.
min_distance = 100 – если игрок ближе указанной дистанции, то монстр не заспавнится (по дефолту 150 метров, но на самом деле это много).
```

### 3.7.2. Спавн предмета в объект

Функция спауна объекта в объект  
=spawn\_object\_in(имя предмета:имя спауна)  
=spawn\_object\_in(wpn\_ak74:esc\_story\_box)

## 3.8. Скрипт logic

**NB:** если хотите заспавнить у прс что-то из вещей из custom data, то описание того, как это делается находится в Общей части в настройке профилей персонажей (только tag supplies писать не надо!)

Скрипт logic управляет переключением схем.

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

Функции, на которые ссылается секция [logic] должны находиться в файлах \gamedata\scripts\xr\_effects.script или \gamedata\scripts\xr\_conditions.script.

В секции должно присутствовать одно из полей:

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.

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 секции текущей активной схемы. Существуют следующие условия переключения:

Список доступных схем перечислен в главе схемы.

**Примечание:** если logic переключает между несколькими одноименными схемами (например несколькими walker), то их можно нумеровать (walker1, walker2) или через @ давать более информативные названия walker@day, walker@alarm и т.д.

on\_actor\_dist\_le = number | scheme - дистанция до игрока <= number  
 on\_actor\_dist\_le\_nvis = number | scheme - дистанция до игрока <= number без проверки на  
 видимость  
 on\_actor\_dist\_ge = number | scheme - если дистанция до игрока > number  
 on\_actor\_dist\_ge\_nvis = number | scheme - если дистанция до игрока > number без проверки на  
 видимость  
 on\_signal = signal | scheme - срабатывает по приходу сигнала signal от текущей активной схемы  
 on\_info = scheme - срабатывает всегда  
 on\_timer = msec | scheme - срабатывает через msec мс после включения схемы  
 on\_game\_timer = sec | scheme - срабатывает через sec секунд игрового времени, после включения  
 схемы  
 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 = npc\_story\_id | restrictor\_name | scheme - если NPC 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  
 on\_actor\_outside = scheme - зона проверяет, находится ли игрок за ее пределами

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

on\_info = {...} %...%  
 on\_info2 = {...} %...%  
 on\_info3 = {...} %...%  
 и так далее до посинения

а также условия для переключения на описанные выше секции.

combat\_ignore\_cond =  
 on\_hit =  
 on\_death =  
 on\_combat =  
 on\_use =

### 3.8.1. Синтаксис скрипта Logic

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 вернул 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: Обратнo из секции nil под скрипты объект вернуть уже невозможно! Учитывайте это, используя

```
ее.
```

### 3.8.2. Вот пример достаточно сложной логики:

```
[logic]
```

```
active = walker
```

```
combat_ignore = combat_ignore
```

```
on_hit = hit
```

```
on_death = death
```

```
[hit]
```

```
on_info = %+alert%
```

```
[death]
```

```
on_info = %+alert +trup3%
```

```
[walker]
```

```
path_walk = walk_svboda3
```

```
path_look = look_svboda3
```

```
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}
```

```
[combat_ignore]
```

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
```

```
combat_ignore = combat_ignore
```

```
[walker]
```

```
path_walk = soldier_walk1
path_look = soldier_look1
combat_ignore_cond = always
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
```

```
[combat_ignore]
```

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).

### 3.9. 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.

#### 3.9.1. 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.

файл `\gamedata\scripts\sr_idle.script`

#### 3.9.2. 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]
```

файл \gamedata\scripts\sr\_no\_weapon.script

### 3.9.3. Секция [sr\_sound],

snd = Перечень имён звуков разделенных запятыми.

type = Типы звуков через запятые. Для удобства введены типы наборов звуков. Т.е., например, чтобы не перечислять каждый раз весь набор звуков скрипа деревянного пола, можно указать тип floor\_wooden.

\*delay = Задержка перед проигрыванием звука в секундах реального времени, по умолчанию 0.

\*idle = Длина периода игнорирования входа в зону после начала последнего проигранного звука. Чтоб, например, завывание было не чаще, чем раз в несколько минут. В секундах игрового времени. По умолчанию 0.

\*rnd = Вероятность (в процентах) того, что звук отыграется. По умолчанию 100.

\*position = Задаёт имя пути, в вершинах которого может отыгаться звук. Есть зарезервированное значение random. Оно означает случайное место в радиусе 15...50 метров от игрока. Если этот параметр не задан, то подразумевается позиция игрока.

\*slide\_velocity = Скорость (м/с) передвижения звука по точкам патрульного пути. По умолчанию - 3

\*slide\_sound\_once = true/false

true - проиграть звук один раз, даже если он не дошел до последней точки пути.

false – если звук закончился, а до последней точки пути не дошел, запустить его ещё раз. По умолчанию false.

\*play\_at\_actor = true/false Заставляет звук играть от позиции актёра постоянно. Если он будет равен true и будет задан путь перемещения звука (или рандом), то мы тупо вылетим.

Предназначение данной схемы: отыграть звук при входе актёра в рестриктор.

**Поддерживается sound\_end.**

Обязательно нужно задать либо snd, либо type. Можно их задать вместе. На базе этих параметров составляется список звуков. При входе актёра в рестриктор отыгрывается случайный звук из этого списка.

Место, из которого может отыгаться звук, задаётся одним из трёх:

- случайное;
- случайная вершина заданного пути;
- позиция игрока.

Пример настроек рестриктора:

```
[logic]
active = sr_sound

[sr_sound]
type = floor_wooden
snd = ambient\wind1, ambient\sparks1
rnd = 50
position = random
idle = 120
delay = 3
```

Есть возможность сделать «скользящий звук». Необходим патрульный путь. Звук начинает отыгрываться с начала пути и перемещается от одной точки пути к другой (по мере их установки на патрульном пути) со скоростью `slide_velocity`.

```
[logic]
active = sr_sound

[sr_sound]
type = random
position = way
slide_velocity = 8
slide_sound_once = true
```

Файл `\gamedata\scripts\sr_sound.script`

### 3.9.4. Секция [sr\_tip]

Предназначение данной схемы – давать игроку сообщение (подсказку) при входе в рестриктор

`name` = Название новости.  
`type` = по умолчанию «news»

Тип новостей: «news» – отсылается как глобальная новость, «tips» - отсылается то имени sender-a

\*`sender` = если тип = «tips», то от sender задаёт условный строковый идентификатор иконки персонажа, от которого якобы пришло сообщение. По умолчанию это иконка торговца.

\*`cond` = Необходимые логические условия, при которых подсказка сработает. По дефолту, сработает при входе в зону.

\*`single` = true/false (по умолчанию false). Если параметр в true, то типс будет выдан только один раз,

Пример настроек рестриктора:

```
[logic]
active = sr_tip

[sr_tip]
name = tips_esc_trader_about_pda
type = tips
```

```
cond = {+infoportion1 -infoportion2 }
```

\*showtime = msec – время в миллисекундах, в течение которого сообщение будет находиться на экране. – **ПОКА НЕ РАБОТАЕТ НОРМАЛЬНО!**

Если необходимо проиграть только 1 раз, а это случается часто, то можно добавить следующую строку:

```
on_actor_inside = nil
```

файл \gamedata\scripts\sr\_tip.script

### 3.9.5. 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%
```

### 3.9.6. Sr\_territory

Занимается эта схема тем, что отлавливает всякие события, происходящие внутри рестриктора.

Пока что она отлавливает только хиты и смерть сталкеров. Пример использования примерно следующий:

```
[logic]
active = sr_territory@outside
```

```
[sr_territory@outside]
on_actor_inside = sr_territory@inside
```

```
[sr_territory@inside]
on_actor_outside = sr_territory@outside
territory_hit = {-bar_dolg_territory_1_hit} %+bar_dolg_territory_1_hit%, {-bar_dolg_territory_2_hit}
%+bar_dolg_territory_2_hit%, {-bar_dolg_territory_3_hit} %+bar_dolg_territory_3_hit%
territory_death = {-bar_dolg_territory_kill} %+bar_dolg_territory_kill%
```

То есть здесь видно, что когда игрок находится внутри рестриктора, то считается количество нанесенных хитов, а также учитывается был ли кто-то убит или нет. Поскольку схема работает только с игроком – то хиты и смерть засчитываются только от игрока.

### 3.9.7. Sr\_mapspot

При входе в рестриктор он сам себя подсвечивает на карте.

Параметры:

hint - id подсказки в string table (обязательный параметр)

location - название типа подсветки (не обязательный параметр, по умолчанию "crlc\_small")

Пример:

```
[logic]
```

```
active = sr_mapspot
```

```
[sr_mapspot]
```

```
hint = "gar_swamp"
```

```
location = crlc_big
```

### 3.9.8. Sr\_particle

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

1) для частицовой системы с путем камеры:

```
[sr_particle]
```

```
name = explosions\campfire_03 -имя частицовой системы
```

```
path = particle_test.anm -имя пути камеры
```

```
mode = 1 (обязательно !!!)
```

```
looped = true/false -флаг зацикленности частиц
```

(обязательно с расширением ANM !!!) Здесь частицы будут молча перемещаться по пути.

2) для частицовой системы с обычным патрульным путем:

```
[sr_particle]
```

```
name = explosions\campfire_03 -имя частицовой системы
```

```
path = part_points -имя патрульного пути
```

```
mode = 2 (обязательно !!!)
```

```
looped = true/false -флаг зацикленности частиц
```

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 частицы. Звук не закливается и играет только один раз.. Результат = частицы отыгрываются во всех вейпоинтах одновременно (или с задержкой см. выше).

При looped=true по окончании проигрывания 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.

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

### 3.9.9. Sr\_sound\_act

Итого, схема, которая играет саунд в голове актера. Всякие там переговоры по ПДА и прочие фейки

```
[sr_sound_act]
snd = ambient\random\new_drone1 --имя звукового файла
*delay = 2000 --задержка перед проигрыванием
*delay_max = 4000 -- между проигрыванием звука будет взят случайный промежуток между
delay и delay_max.
*on_signal = sound_end | nil --по сигналу можно перейти в другую секцию.
theme = <имя темы из ph_sound_themes>
* stereo = true/false (по умолчанию false). При установке этого параметра к файлу, который
задан параметром snd или в звуковой теме будут добавляться (автоматически) суффиксы _l и _r для
загрузки левого и правого каналов и, соответственно, вся эта фигня будет играть.
```

Если указывается тема, то звук будет играть за циклено, случайным образом выбирая один из звуков прописанных в теме, если указывается звук, то он отыгрывается один раз. Схема поддерживает кондлист.

### 3.9.10 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 декриментирующий(dec).

Если поле не задано - счетчик будет инкрементирующий

start\_value - начальное значение счетчика в РЕАЛЬНЫХ миллисекундах. Для декриментирующих 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%
```

### 3.9.11. 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
```

### 3.9.12. Sr\_teleport

Собственно, телепорт. Настраиваются следующим образом:

```
[logic]
active = sr_teleport
```

```
[sr_teleport]
timeout = 0
```

```
point1 = point1
look1 = look1
prob1 = 10
```

```
point2 = point2
look2 = look2
prob2 = 20
```

где:

timeout - задержка в срабатывании телепорта в миллисекундах.

point - одноточечный патрульный путь куда переместить

look - одноточечный патрульный путь куда повернуть.

Далее идут настройки точек назначения с удельными весами. То есть в перечисленном выше примере вероятность телепортнуться во вторую точку в два раза выше, чем в первую. Максимальное количество точек назначения - 10. Телепорты необходимо ставить совместно с особой аномальной зоной, которую сейчас делает Проф. Зона добавит визуализацию и создаст эффект втягивания.

### 3.9.13. Sr\_sleep и настройка снов.

Появилась возможность задавать зоны сна.

```
[sr_sleep]
*cond = <condlist>
```

\*type = nightmare/normal/happy/all - Задаёт тип сна разрешённый в данной зоне (по умолчанию all).

Влияет (группирует) только на несценарные сны.

\*dream\_prob = <число от 0 до 100> - вероятность просмотра несценарных сновидений в данной зоне (по умолчанию 80). В противном случае будет только чёрный экран.

Необязательное поле cond задаёт условие(я), при котором в этой зоне можно спать. Сейчас производится индикация зон, где разрешён сон. В левом нижнем углу отображается маленькая иконка легких при входе в такую зону. Вероятно, позже будет изменена на другую.

Сновидения теперь делятся на сценарные и обычные. Сценарные сновидения отыгрываются один раз при выполнении необходимых условий. Обычные сновидения проигрываются, если нет сценарных или ни одно условие выполнения сценарных не сработало. Можно задавать вероятность отыгрыша обычных сновидений в целом, а также задавать вероятность срабатывания каждого конкретного сновидения в отдельности. Обычным сновидениям можно задавать тип и потом ограничивать по нему сны воспроизводимые в sr\_sleep.

В файле misc\dream.ltx задаются настройки снов.

Секция videos.

Полями задаются пути к видеофайлам со снами.

Секция dreams. Поля:

regular\_probability = <число от 0 до 100> - вероятность проигрывания обычных сновидений в целом

regular - список секций с настройками для обычных сновидений

scene - список секций с настройками для сценарных сновидений

Настройки обычных сновидений:

dream - имя поля из секции videos

probability = <число больше 0> - чем больше, тем больше вероятность проигрывания сна.

type = nightmare/normal/happy - тип сна.

Настройки сценарных сновидений:

dream - имя поля из секции videos

cond = <condlist> - условия срабатывания

to\_regular = <вероятность,тип> - необязательное поле. Дает возможность переводить сценарный сон после первого отыгрыша в разряд обычных. <вероятность, тип> аналогичны probability и type из настроек обычных сновидений соответственно.

### 3.9.14. Sr\_cutscene

Эта схема предназначена для проведения анимации камеры с некоторым эффектом (pp\_effector). Последовательность действий, осуществляемых схемой, состоит из мгновенного перемещения игрока в начало пути point и ориентации его взгляда на начало пути look, потери управления игроком и начала анимации камеры cam\_effector по завершении которой игрок вновь получает управление.

[sr\_cutscene]

point = <имя пути> - путь в первую точку которого переносится игрок

look = <имя пути> - путь в первую точку которого смотрит игрок

\*pp\_effector = <имя файла с эффектом> - файл, расположенный в папке gamedata\anim\ и содержащий эффект (имя файла пишется без расширения)

cam\_effector = <имя файла с анимацией камеры> - файл, расположенный в папке gamedata\anim\camera\_effects\ и содержащий анимацию камеры (имя файла пишется без расширения)

### 3.10. Набор дополнительных настроек логики у разных объектов.

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

#### 3.10.1. 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` проигрывается `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
```

файл `\gamedata\scripts\ph_door.script`

#### 3.10.2. 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

Файл \Gamedata\scripts\ph\_button.script

\*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%
```

### 3.10.3. Схема работы прожектора:

В точках look пути, в которые смотрит прожекторщик, нужно прописать sl=имя\_прожектора

Например  
wp00|sl=esc\_sl1

Тогда при повороте в эту точку персонаж повернет в нее и прожектор.

### 3.10.4. Кодовые замки:

При введении указанного кода выдает инфопоршн

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

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

Файл: \gamedata\scripts\ph\_code.script

### 3.10.5. 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
```

Файл: \gamedata\scripts\ph\_gate.script

### 3.10.6. Ph\_sound

Прописывается у физического объекта какие звуки он выдает (изначально планировался как матюгальник).

```
[ph_sound]  
snd = имя темы из файла sound_theme.script из таблицы ph_snd_themes  
*looped = true/false зацикленное воспроизведение звука (default - false)  
*min_idle = минимальное время простоя перед включением звука (мс)  
*max_idle = максимальное время простоя перед включением звука (мс)  
*random = true/false (def - false). Если = true, то из темы будет выбран рандомный звук и таким образом звуки будут играть до посинения
```

**NB! Если мы задаем random = true и looped = true, то версия сыпется**

Также поддерживается кондлист.

Данная схема работает через задницу, поэтому зацикленный звук будет продолжать отыгрываться, даже если объект уходит в nil. В связи с этим надо создавать новую секцию, которая бы отыгрывала одиночный короткий звук, после которого (поскольку он будет точно также играть раз за разом) ставим on\_signal = sound\_end| nil

Пример подобной извращенной логики:

```
[logic]  
active = ph_sound
```

```
[ph_sound]  
snd = gar_seryi_shooting  
looped = true  
max_idle = 5000  
on_actor_in_zone = gar_seryi_factory| ph_sound@end
```

```
[ph_sound@end]  
snd = gar_seryi_shooting_2  
looped = false  
on_signal = sound_end| nil
```

Кроме того специфическим образом создается звуковая схема.

В `sound_theme.script` в начале файла есть секция `ph_themes` в которой и описываются темы для физ объектов.

Например:

```
ph_snd_themes["gar_seryi_shooting"] =  
{[[characters_voice\human_01\scenario\garbage\distance_shooting]]}
```

Кроме того (незадекларированная фича) `ph_sound` можно вешать на рестрикторы. Но за правильность работы в таком случае никто ответственности не несет.

Файл: `\gamedata\scripts\ph_sound.script`

### 3.10.7. 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.

### 3.10.8. Ph\_on\_death

Схема для отслеживания разрушения физического объекта и выдавания по такому случаю различных эффектов

Пример:

```
[logic]  
active = ph_on_death
```

```
[ph_on_death]  
on_info = %эффекты%
```

Юзать исключительно с разрушаемыми физ. Объектами

### 3.10.9. Ph\_car

Настройка возможности игроку управлять машиной.

секция: `[ph_car]`

поле: `usable = <condlist>`

`usable` - кондлист возвращающий `true` (по умолчанию) или `false`.

Пример:

```
[logic]  
active = ph_car
```

```
[ph_car]  
usable = {+val_actor_has_car_key}
```

На основе этой схемы можно сделать машину, которая взведется только если у актера есть ключ именно от нее.

### 3.10.10. 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]
```

### 3.10.11. Ph\_oscillate

Схема предназначена для плавного раскачивания физики (лампы, висящие 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.

## 3.11. Смарттеррейны и гулаги.

### 3.11.1. Смарттеррейн.

Под смарттеррейном мы понимаем зону, зайдя в которую, сталкер на некоторое время попадает под гулаг и начинает выполнять работу, предусмотренную этим гулагом. После некоторого времени он выходит из-под гулага и ходит свободно.

Как поставить smart terrain?

Для всех smart terrain нужно:

- 1) Поставить smart terrain с необходимым shape. Большой shape не рекомендуется (размер влияет на производительность).
- 2) В его custom data прописать настройки.
- 3) Расставить пути для соответствующих схем поведения.

Параметры custom data:

```
[gulag1]
```

```
type = тип гулага
```

```
capacity = макс. вместимость в людях
```

```
*offline = может ли гулаг образоваться в offline (true(по дефолту)/false)
```

```
*squad = squad, который будет проставлен всем сталкерам под гулагом (№ уровня)
```

```
*groups = набор group через запятые
```

```
*stay = min, max время пребывания прс под smart_terrain (по умолчанию – навсегда)
```

```
*idle = min, max время бездействия smart_terrain после ухода последнего прс
```

\*cond = список условий, которые необходимы для создания гулага {+info -info =func !func} – если условие не выполняется, то гулаг распускается, а все его подопечные начинают управляться прописанной в custom\_data логикой.

Указывать тип гулага нужно без кавычек.

Если не задан squad или groups, то соответствующие свойства сталкеров не будут изменяться.

Все времена задаются в часах игрового времени и могут быть дробными.

Пути:

Имена путей для схем поведения всегда должны начинаться с имени данного smart terrain.

Например, esc\_smart\_ambush\_vagon\_sleep.

Если пути для smart terrain на нескольких человек (campers, walkers), то их имена должны заканчиваться всегда на цифру (esc\_smart\_ambush\_vagon\_walk1, esc\_smart\_ambush\_vagon\_walk2)

Гулагов под одним smart terrain может быть несколько. Их можно настраивать в секциях [gulag2], [gulag3] и т.д. При входе сталкера под smart terrain будет случайно выбран один из доступных на данный момент гулагов.

### 3.11.1.1. Стандартные типы смарттеррейнов.

Если нужно, чтоб сталкер не захватывался, допишите ему в custom data следующую строку:

```
[smart_terrains]
none = true
```

Если сталкер уже под каким-то smart terrain, то остальные smart terrain он будет игнорировать.

#### **campers**

Кемперы. custom data:

```
[gulag1]
type = campers
capacity = от 1 до 3
```

Пути:

```
camper_walk1, camper_look1
camper_walk2, camper_look2
camper_walk3, camper_look3
```

#### **walkers**

Ходячие. На базе этого можно сделать поиск, обыск и куча всего.

custom data:

```
[gulag1]
type = walkers
capacity = от 1 до 3
```

Пути:

```
walker_walk1, walker_look1
walker_walk2, walker_look2
walker_walk3, walker_look3
```

## search

Ходячие. На базе этого можно сделать поиск, обыск и куча всего.

custom data:

```
[gulag1]
type = search
capacity = 1
```

Пути:

search\_walk, search\_look

Схема следующая:

1. Персонаж ходит по точкам, смотрит по сторонам
2. В определенных точках останавливается и что-то высматривает (caution, search, hide)
3. При этом говорит определенные реплики (...)

## rest

Отдых. Сталкер по очереди to sleeper, to walker, to rest(ест еду, пьёт водку).

custom data:

```
[gulag1]
type = rest
capacity = 1
```

Пути:

rest – путь из двух вершинок (возможно из 1). В одной сидит, в другую смотрит.  
sleep - путь из двух вершинок (возможно из 1). В одной спит, в другую смотрит.  
rest\_walk, rest\_look

### 3.11.2. Гулаги.

Гулаг - средство объединения нескольких сталкеров под централизованным управлением.

Основные особенности:

А) Есть список работ гулага. Работа - настроенная схема поведения (или цепочка схем поведения);

Б) Работы имеют приоритеты;

В) Гулаг назначает на работы сталкеров входящих в гулаг, начиная с работ с наивысшим приоритетом;

Г) Гулаг имеет состояния. Каждое состояние характеризуется своим набором работ, отличным от набора работ в любом другом состоянии гулага.

Гулаг создается следующим образом:

1. Необходимо четко определить набор состояний гулага: день, ночь, спокойное, при тревоге и так далее. Для простых гулагов достаточно одного состояния, для крутых и сложных – желательно разные. Это придает разнообразия и смотрится лучше.

2. Определяем максимальное количество людей, которым гулаг может управлять. То есть определяем вместимость гулага. Она должна быть такой, чтобы в любом состоянии гулага гарантированно нашлось занятие для каждого человека.

3. Для каждого состояния гулага определяется набор работ. Эти работы могут быть как активного плана (часовой, патруль, и так далее), так и пассивного плана (сидят вокруг костра, спят).

Каждая работа имеет свой приоритет. Соответственно пассивные работы должны иметь меньший приоритет.

4. Ставится в редакторе количество людей, которые должны быть под гулагом, и накрываются зонкой smart\_terrain (**источник ошибок – иногда ставят space\_restrictor**). Зонке нужно давать осмысленное название. Это же название будет являться префиксом к названиюм всех патрульных путей, относящихся к этому же гулагу. Например если вы назвали зонку esc\_blockpost, то все патрульные пути должны начинаться с этого префикса, например esc\_blockpost\_guard\_walk. В custom\_data зоны необходимо прописать настройку гулага.

```
[gulag1]
type = тип гулага
capacity = макс. вместимость в людях
*offline = может ли гулаг образоваться в offline (true/false)
*squad = squad, который будет предоставлен всем сталкерам под гулагом
*groups = набор group через запятые
*stay = min, max время пребывания прс под smart_terrain
*idle = min, max время бездействия smart_terrain после ухода последнего прс
*cond = список условий, которые необходимы для создания гулага {+info -info =func !func} –
если условие не выполняется, то гулаг распускается, а все его подопечные начинают управляться
прописанной в custom_data логикой.
```

\*respawn = имя респауна (вызывает респаунер с заданным именем каждый раз, когда кто-то из самрттеррейна заступает на работу)

Capacity нужно задавать всегда. Она может быть равна или меньше числа работ.

Указывать тип гулага нужно без кавычек.

Поле offline можно задать, чтоб гулаг не образовывался в офлайн. Т.е. существовать в офлайн он может, а образовываться – нет.

Если не задан squad или groups, то соответствующие свойства сталкеров не будут изменяться.

Все времена задаются в часах игрового времени и могут быть дробными.

5. В скрипте \gamedata\scripts\gulag\_название\_уровня.script необходимо прописать условия, при которых сталкеры берутся под конкретный гулаг. В функцию checkNPC необходимо прописать условие:

```
if gulag_type == "gar_dolg" then
    return npc_community == "dolg"
end
```

В эту функцию пока передается два параметра, тип гулага и комьюнити персонажа. В данном случае под гулаг с типом gar\_dolg будут приниматься все персонажи, относящиеся к группировке Долг.

6. В файле \gamedata\scripts\gulag\_название\_уровня.script необходимо описать переключение состояний гулага.

```
function loadStates(gname, type)
```

в нее передается имя зонки и тип гулага. Состояние гулага описывается в виде функции, возвращающей номер состояния гулага. Например:

```
if type == "gar_maniac" then
    return function(gulag)
        if level.get_time_hours() >= 7 and level.get_time_hours() <= 22 then
            return 0 -- день
        else
            return 1 -- ночь
        end
    end
end
```

end

В данном случае если сейчас между 7 и 22 часов, то гулаг находится в дневном состоянии, иначе в ночном.

8. В файле \gamedata\scripts\gulag\_название\_уровня.script необходимо описать должности гулага. В функции loadJob загружаются все допустимые работы. В саму функцию передаются следующие параметры:

function loadJob(sj, gname, type, squad, groups)

sj – сама табличка работ гулагов,

gname – имя нашей зонки смар-тиррейна. Оно используется как префикс.

Type – тип гулага

Squad, groups – таблички сквадов и групп, если нам нужно переопределять родные группы сталкеров на какие либо другие. В каждой работе можно указать какой сквад и группа сетится сталкеру при установке на работу.

Примерное описание работ гулага:

Данный гулаг описывает поведение только одного человека, обычно их гораздо больше.

Данный человек в нулевом состоянии(день) делает одну работу, в первом состоянии(ночь) делает другую работу.

```
--' Garbage maniac
if type == "gar_maniac" then
    t = { section = "logic@gar_maniac_camper",
        idle = 0,
        prior = 5, state = {0},
        squad = squad, groups = groups[1],
        in_rest = "", out_rest = "",
        info_rest = ""
    }
    table.insert(sj, t)
    t = { section = "logic@gar_maniac_sleeper",
        idle = 0,
        prior = 5, state = {1},
        squad = squad, groups = groups[1],
        in_rest = "", out_rest = "",
        info_rest = ""
    }
    table.insert(sj, t)
end
```

Описание полей:

Idle – пауза между повторным выполнением одного и того же задания. В данном случае паузы нет. Обычно пауза ставится на патруль.

Prior – приоритет задания. Сперва сталкеры занимают более приоритетные задания. Чем больше число, тем выше приоритет.

In\_rest, out\_rest - рестрикторы, которые устанавливаются персонажу на данное задание.

Section – секция в \gamedata\config\misc\gulag\_название\_уровня.ltx, где указываются реальные настройки схемы поведения, которая соответствует текущей работе.

Group сталкера будет выбран из массива groups, который задан в custom data. Массив индексируется начиная с 1.

Info\_rest – задает ся имя рестриктора и все денжеры снаружи этого рестриктора не попадают внутрь для человека, находящегося на этой работе

Также в описании работы может быть указаны дополнительные условия, при которых сталкер может занять данную работу. Например:

```
predicate = function(obj)
    return obj:profile_name() == "soldier_commander"
end
```

то есть данную работу сможет выполнять лишь персонаж с профилем soldier\_commander.

9. В \gamedata\config\misc\gulag\_название\_уровня.ltx необходимо указать, какие схемы поведения соответствуют той или иной работе. Например в случае с вышерассмотренным гулагом gar\_maniac:

```
;-----
;-- GARBAGE MANIAC
;-----
[logic@gar_maniac_camper]
active = camper@gar_maniac_camper

[camper@gar_maniac_camper]
path_walk = walk1
path_look = look1

[logic@gar_maniac_sleeper]
active = sleeper@gar_maniac_sleeper

[sleeper@gar_maniac_sleeper]
path_main = sleep
wakeable = true
```

Настройка здесь соответствует настройке в обычной кастом дате сталкера, с разницей:

- 1) пути следует указывать без префикса. То есть если зонка носила название gar\_maniac, то пути следует на уровне ставить с названием gar\_maniac\_walk1, однако в gamedata\config\misc\gulag\_название\_уровня.ltx следует указывать только walk1.
- 2) в именах секций схем поведения после @ добавлять название гулага и, возможно, дополнительные сведения (например, walker2@rad\_antena\_gate)
- 3) в именах секций logic для каждой работы добавлять после @ имя гулага, дополнительные сведения и имя секции активной схемы поведения (например, logic@rad\_antena\_gate\_walker2).

В работах для гулагов поля leader больше нет. Есть поле dependent. Работа может быть занята только тогда, когда работа с именем dependent уже занята. Например, follower может быть назначен только тогда, когда уже кто-то назначен на работу лидера (имя работы лидера теперь в поле dependent). Естественно, что приоритет работ, от которых зависят другие, должен быть больше чем у них.

### 3.11.3. Новые особенности смарттеррейнов

#### Возможности нового смарттеррейна (СТ):

- 1) Не держит сталкеров постоянно в онлайн. Работает стандартный онлайн-радиус.
- 2) Сталкеры идут на ближайшие работы.
- 3) На места работ сталкеры идут независимо от того, в онлайн они или в оффлайне.
- 4) СТ в офлайне работает так же, как и в онлайн: выполняет переключение своих состояний, перераспределение работ.
- 5) Сталкерам можно прописать, при каких условиях в какие СТ они могут идти. (см. ниже) Если сталкер попал в СТ, то онбудет находится в нём, пока не истечёт время и выполняется условие.

- 6) Работы могут находиться на разных уровнях.
- 7) Скриптовая зона СТ теперь не используется для захвата персонажей.
- 8) Симуляция заключается в миграции персонажей между разными СТ.

### **Что нужно переделать:**

- 1) Персонажи могут быть двух типов: либо для СТ, либо для самостоятельной работы под логикой из custom data. У первых логики в custom data не должно быть. У вторых должно быть прописано, что они не хотят ни в один СТ. (см ниже)
- 2) Нельзя под СТ отправлять сталкеров в nil. Вместо nil дайте им пути. Например, walker-ы в рестрикторе вместо nil в рестрикторе. (есть abort на такой случай)
- 3) Всех участников созданных сцен поставьте рядом с местами работ, а не в кучу. Так им не придётся полчаса разбредаться по местам работ: они сразу позанимают ближайшие. В custom data им пропишите, что до окончания сцены они могут быть только в этом СТ. (см. ниже)
- 4) Незначительно переделать функции predicate() и функции переключения состояния СТ. (см. ниже)
- 5) Проследите, чтоб под СТ в логиках в поле active было прописано только имя секции и ничего больше (никаких там процентов и фигурных скобок). Для персонажей не предназначенных под СТ это не играет роли.
- 6) Переименуйте в custom data СТ секцию [gulag1] в секцию [smart\_terrain].

### **----- Настройки: -----**

#### **---- Разрешения персонажам идти в определённые СТ ----**

Разрешения персонажам идти в определённые СТ задаются в custom data секцией [smart\_terrains]. В ней можно задавать пары "имя\_СТ = condlist". Пример:

```
[smart_terrains]
strn_1 = условие1
strn_2 = условие2
```

Если для какого-то smart\_terrain условие выполнилось, он называется эксклюзивным.

Если у объекта появился хоть один эксклюзивный smart terrain, то он будет согласен идти только в него.

Если не появилось ни одного эксклюзивного, то он согласен идти в любой.

Есть зарезервированное сочетание "none=true". Если оно указано, то персонаж никогда не пойдёт ни в один СТ. Такой персонаж будет работать только под своей логикой.

Также можно задать, кого принимает СТ. В дополнение к старому механизму (функции checkNpc() в файлах gulag\_\*.script) можно в custom data СТ написать:

```
communities = группировка1, группировка2, ...
```

Если это поле не задано, то проверяется старым механизмом. Если задано, то под СТ возьмутся только персонажи указанных группировок (учтите, старый механизм тоже вызовется).

#### **---- Изменение функций predicate() ----**

В эти функции вместо game\_object будет передаваться табличка с информацией о персонаже. Там есть поля:

```
name
community
class_id
story_id
```

profile\_name

Если нужно, чтобы работа занималась только снайперами, то в предикате нужно писать:

```
predicate = function(npc_info)
    return npc_info.is_sniper == true
end
```

#### ---- Изменение функций переключения состояния СТ ----

Обращайтесь индивидуально. Все переделки связаны с работой этой функции в офлайне. Например, таблица gulag.Object[] не содержит game\_object-ы, если персонаж в офлайне и т.п.

---- Состояния работ online/offline

```
t = { section = "logic@ЧЧЧЧЧЧЧЧ",
      idle = 0,
      prior = 5, state = {0}, squad = squad, group = groups[1],
      online = true,
      in_rest = "", out_rest = ""
    }
table.insert(sj, t)
```

Варианты задания этого поля

online = true - на этой работе персонаж всегда в онлайне,

online = false - на этой работе персонаж всегда в офлайне,

online не задано - на этой работе персонаж может прыгать онлайн<->офлайн по своему усмотрению.

### 3.11.3.1. Более доступное описание новых смарттеррейнов

Теперь о смарттеррейнов для дизайнеров, то есть не на LUA, а по-русски.

Для того, чтобы пренести смарттеррейн на новую схему, делаем следующее:

1. Пишем в кастом дате где [gulag1] -> [smart\_terrain]

2. В кастом дате товарищей по смарттеррейну пишем

```
[smart_terrains]
```

sar\_monolith\_sklad(название гулага) = {кондлист} - если только в 1 смарттеррейн сталкер сможет прийти, то пишем true.

Если этот товарищ не должен работать под смарттеррейнами, то пишем ему в кастом дату.

```
[smart_terrains]
```

```
none = true
```

## 3.12. Логика вертолёт

**Общие сведения:**

Вертолёт работает на «логике».

На вертолёт реагируют аномалии.

Вертолёт не обрабатывает столкновения с геометрией и физикой пока он не сбит.

Попадания в область кабины, где сидит первый пилот, в десятки раз более болезненны для вертолёт.

У вертолёт есть универсальная боевая схема на манер сталкеров.

Пилоты вертолёт реагируют репликами на события: хит, видит врага, поврежден (задымился), падает.

### 3.12.1. Схема heli\_move:

#### Общие сведения:

Позволяет летать вертолёту по патрульному пути, регулировать максимальную скорость, смотреть в нужную точку. Скорость между точками рассчитывается автоматически, она может быть меньше максимальной, но никогда ее не превысит.

Для схемы должен быть задан path\_move – путь, по которому будет летать вертолёт. Он может содержать одну вершину, если нужно, чтоб вертолёт висел на месте.

Можно (но не обязательно) задать path\_look – точка, куда вертолет может смотреть, можно задавать асторг – тогда будет смотреть на игрока.

Вершины этих путей могут быть поставлены где угодно в пределах ограничивающего бокса уровня. Они **не** зависят от ai-nodes.

По пути вертолёт летает без учёта связей между вершинами. Он летает от вершины к вершине в порядке возрастания их номера (т.е. в порядке, в котором их поставили на уровень).

Вертолёт старается летать точно по вершинам пути. При желании можно сделать ювелирный пролёт под мостом.

Вертолёт старается летать как можно быстрее. Пояснение: если ему задать, что в следующей вершине пути он должен иметь скорость 10 м/с, а его максимальная скорость установлена в 30 м/с, то он не станет сразу лететь 10 м/с. Он сначала будет разгоняться вплоть до 30 м/с и только на подлёте к целевой вершине начнёт тормозить с расчётом прибыть в неё имея 10 м/с.

#### Настройки:

Обязательные настройки:

\*path\_move = путь, по которому будет летать вертолёт, задается вейпоинтами.

\*max\_velocity = km/h – максимально допустимая скорость, если возможно, он будет стремиться к ней. Если на пути вертолет промахивается по точкам – следует уменьшить max\_velocity.

Задавать не обязательно:

\*enemy = nil/actor/StoryID - враг

\*fire\_point = – точка, обстрел которой вертолет будет делать, задается вейпоинтом.

\*min\_mgun\_attack\_dist = ;m мин расстояние при котором можно исп пулемет

\*max\_mgun\_attack\_dist = ;m макс расстояние при котором можно исп пулемет

\*min\_rocket\_attack\_dist = ;m мин расстояние при котором можно исп ракеты

\*max\_rocket\_attack\_dist = ;m макс расстояние при котором можно исп ракеты

\*use\_rocket = false/true выкл./вкл. стрельбу ракетами, если не заданно будет считаться true

\*use\_mgun = false/true выкл./вкл. стрельбу с пулемета, если не заданно будет считаться true

\*engine\_sound = true/false (по умолчанию true). Вкл/выкл звук двигателя вертолёта.

\*upd\_vis = число, - время, в секундах, через которое проверяется видимость врага.

\*stop\_fire = true/false (по умолчанию false). Если увидит игрока, остановиться чтобы смотреть на игрока с позиции и обстреляет, если игрок – враг.

\*show\_health = true/false (по умолчанию false). Отображается индикатор жизни.

\*fire\_trail = true/false (по умолчанию false). Вкл/выкл стрельбу линией(не в точку).

\*invulnerable = true/false (по умолчанию false). Неуязвимость. Если true, вертолёт игнорирует все хиты.

\*immortal = true/false (по умолчанию false). Бессмертие. Если true, вертолёт получает повреждения, но не умирает.

\*mute = true/false (по умолчанию false). Отключает универсальные реплики пилотов вертолета.

### 3.12.2. Универсальная боевая схема:

#### Общие сведения:

В универсальной боевой схеме вертолёт не привязан к путям.

Вертолёт не видит никого. Узнать о враге вертолёт может только при получении хита или из параметра в custom data.

Вертолёт стреляет по врагу, если видит его. Если не видит – ищет, облетая вокруг точки, где последний раз видел. Если долго не видит врага – забывает его. Если врага задали принудительно из текущей секции схемы поведения, то он не забудет его, пока находится в этой секции.

#### Настройки:

Отдельной секции для этой схемы поведения нет. Поэтому настройки производятся в секции текущей схемы поведения:

combat\_ignore = true/false

true означает игнорирование получения хита. Т.е. вертолёт не будет пытаться «отомстить» тому, от кого он получил хит.

combat\_enemy = nil/actor/StoryID

С помощью этого параметра можно задать вертолёту конкретного врага. nil – нету врага; actor – игрок; SID – числовое story id врага.

combat\_use\_rocket = true/false

Можно ли вертолёту пользоваться ракетами.

combat\_use\_mgun = true/false

Можно ли вертолёту пользоваться пулемётом.

combat\_velocity = <число>

Скорость, с которой вертолет будет делать боевые заходы

combat\_safe\_altitude = <число>

Высота, относительно самой высокой точки геометрии на уровне ниже которой вертолет не будет опускаться в боевой схеме (может быть отрицательным)

К вертолёту подключена схема xr\_hit. Работает как у сталкеров. В xr\_effects есть группа функций для работы с вертолётom из его custom data:

heli\_set\_enemy\_actor - сделать актёра врагом вертолёту

heli\_start\_flame - поджечь вертолёт

heli\_die - убить вертолёт

combat\_velocity = - боевая скорость в этой секции указывается в м/с

combat\_safe\_altitude = - высота боевая в метрах, может принимать отрицательные значения

combat\_use\_rocket = - true/false использовать ли ракеты в этой секции

combat\_use\_mgun = - true/false использовать ли пулемет в этой секции

### 3.13. Meet\_manager

Синтаксис:

```
[logic]
meet = meet
```

```
[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
use                  = self
use_wpn              = false
zone                 = name| state@sound
meet_dialog          = dialog_id
synpairs             = state@sound|state@sound
abuse                 = true/false
```

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% )
```

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 xr\_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_weap
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`

### 3.14. Отметки на минимэпе

Появилась возможность не показывать сталкеров на минимэпе и на карте (прятать синие и красные точки). Для этого в секции логики или в текущей схеме указываем параметр:

```
[camper]
show_spot = false (будучи в этой секции сталкер не показывается на карте)
```

```
[walker]
show_spot = {+info1} false
```

Сталкер не будет показываться, если у игрока есть инфопоршн info1 и т.д.

### 3.15. Передача параметров в функции.

Ниже перечислен набор функций к которым можно обращаться из кастом даты и при этом передавать в них переменные.

**NB!** Во всех функциях `xr_conditions` и `xr_effects`, которые обращались к гулагам по имени, теперь можно использовать как имя, так и `story_id`. Причем если мы указываем имя, то использовать функцию можно только, когда гулаг находится в онлайн, а если мы вешаем на самрттеррейн `story_id`, то можем обращаться к гулагу и в оффлайне.

Описание функций с параметрами присутствующих в `xr_conditions` и `xr_effects`.

-----  
xr\_conditions:

**fighting\_dist\_ge(p)** – универсальная функция для `combat_ignore`, проверка расстояния для игрока (в метрах)

**distance\_to\_obj\_le(sid:dist)** - проверка дистанции до объекта заданного `story_id`.

Можно использовать, например, в секции follower для определения того, что сталкер подошел на нужную дистанцию к лидеру и переключать в другую секцию (лидер при этом стоит где-то в ремарке). Эта ситуация возникает, когда после боя надо подогнать одного сталкера к другому, а ихних позиций мы не знаем. Если используется в секции follower, то dist надо ставить большим distance фолловера, поскольку если поставить их одинаковыми, то данная функция не всегда будет срабатывать.

**health\_le(health)** - проверка того, что здоровье npc  $\leq$  health

**heli\_health\_le(health)** - аналогично предыдущему, только для вертолета.

**enemy\_group(group1:group2:...)** - Проверка на принадлежность врага к одной из групп (правильность работы пока не проверялась)

**hitted\_by(sid1:sid2:...)** - Проверка того, что удар был нанесен кем-то из npc, указанных в списке. npc задаются с помощью story\_id. Функцию удобно использовать в секции hit.

Пример:

[hit]

on\_info = {=hitted\_by(407:408)} %+val\_escort\_combat%

**killed\_by(sid1:sid2:...)** - Аналогично предыдущему, но для случая смерти npc. Используется в секции death.

**is\_alive(sid)**

**is\_alive\_one(sid1:sid2:...)**

**is\_alive\_all(sid1:sid2:...)** - проверка того, что один, один из нескольких или все из списка соответственно npc, заданные по story\_id живы

**is\_dead(sid)**

**is\_dead\_one(sid1:sid2:...)**

**is\_dead\_all(sid1:sid2:...)** - аналогично предыдущему, только проверка на "мертвость".

**check\_fighting(sid1:sid2:...)** - Проверка того, не является ли кто-то из перечисленных (с помощью story\_id) npc врагом даного. Как правило используется в combat\_ignore\_cond.

**gulag\_empty(gulag\_name)** - проверка того, что гулаг пуст или вообще не существует.

**gulag\_population\_le(gulag\_name, num)** - проверка того, что количество народу в гулаге  $\leq$  num

**gulag\_casualties\_ge(gulag\_name:num)** – проверка того, что гулаг понес потери  $\Rightarrow$  num

**NB! Потери гулага не обнуляются, так что с этой функцией работать аккуратно.**

**signal(строка)** – проверяет, установлен ли у данного НПС в текущей схеме указанный сигнал

-----  
xr\_effects:

**heli\_set\_enemy(story\_id)** – сделать npc с указанным story\_id врагом вертолету. В одной секции можно задавать только 1 врага.

**set\_gulag\_enemy\_actor(gulag\_name)** – сделать актера врагом для данного гулага

**hit\_npc(direction:bone:power:impulse:reverse=false)** - нанести хит по npc. Параметры:

direction - если строка, то считается, что это имя пути и в сторону первой точки производится толчек. Если же это число, то оно рассматривается как story\_id персонажа от которого должен поступить хит.

bone - строка. Имя кости, по которой наносится удар.

power - сила удара

impulse - импульс

reverse (true/false) - изменение направления удара на противоположное. по умолчанию false.

Пример:

[death]

```
on_info = {=killed_by(404)} %=hit_npc(404:bip01_spine1:100:2000)%, {=killed_by(405)}
%=hit_npc(405:bip01_spine1:100:2000)%
```

**set\_friends(sid1:sid2:...)**

**set\_enemies(sid1:sid2:...)** - установить друзьями/врагами данного прс и указанных в списке по story\_id.

**play\_snd(snd\_name:delay=0)** - играть звук в голове актёра.

snd\_name - путь к звуку относительно папки sounds

delay - задержка перед проигрыванием. По умолчанию 0 – проигрываем сразу.

**play\_snd\_now (sid:snd\_name)** – играть звук от указанного объекта

\*звук играет об объекта с указанным story id, без задержки с громкостью 1. Указывается не имя звуковой схемы, а имя файла

**hit\_obj(sid, bone, power, impulse, hit\_src=npс:position())**

Дать объекту, заданому story\_id, хит. Отличается тем, что может прописываться в любой кастом дате. Параметры: actor, npc, p[sid,bone,power,impulse,hit\_src=npс:position()]

1. sid - story\_id объекта, по которому наносится хит.

2. bone - строка. Имя кости, по которой наносится удар.

3. power - сила удара

4. impulse - импульс

5. hit\_src (необязательный параметр) - точка (waypoint), из которой по объекту наносится хит.

Если не задано, то берется позиция объекта, из которого была вызвана данная функция.

**actor\_has\_item(section)**

Проверка на наличие у игрока соответствующего предмета. Проверка проходит по секции в ltx

**Функции для работы с HUD'ом.**

**disable\_ui\_elements(...), enable\_ui\_elements(...)** - отключение/включение элементов HUD'a.

Параметры:

-- weapon - спрятать/показать руки с оружием

-- input - отключить/включить клавиатуру

-- hud - спрятать/показать индикаторы на экране

-- all - отключить/включить все элементы

Пример:

```
on_info = %=disable_ui_elements(weapon:input)%
```

Есть также сокращенные варианты:

**disable\_ui, enable\_ui** (вызываются без скобок и параметров).

Аналогичны вызовам disable\_ui\_elements(all), enable\_ui\_elements(all) соответственно.

Пример:

```
on_info = %=enable_ui%
```

### **Функция запуска camera\_effector'a.**

```
run_cam_effector(имя_файла)
```

имя\_файла (указывается без расширения) - это имя анимационного файла (с расширением anm)

из папки S:\GameData\anim\camera\_effects\.

Пример:

```
on_info = %=run_cam_effector(prison_0)%
```

### **Функция запуска постпроцесса.**

В связи с изменением процесса создания постпроцессов были внесены изменения в их запуск. Теперь есть 2 функции для работы с постпроцессами:

```
run_postprocess(file_name:id:loop) - запуск постпроцесса.
```

-- file\_name - имя файла постпроцесса (без расширения) из папки s:\gamedata\anim.

Указывается без расширения.

-- id - номер постпроцесса. Задается опционально. Используется в stop\_postprocess.

-- loop - (true/false) определяет зацикленность постпроцесса. Опциональный параметр. По умолчанию false.

```
stop_postprocess(id) - принудительная остановка постпроцесса.
```

-- id - номер постпроцесса заданный в run\_postprocess.

### **Функция выброса содержимого инвентаря актера в определенную точку.**

```
drop_actor_inventory(имя_пути)
```

выбрасываем все предметы из инвентаря актера в первую точку заданного пути.

Пример:

```
on_info = %=drop_actor_inventory(drop_point)%
```

## **3.16. Настройка звуковых групп.**

Новый принцип создания звуковых групп:

1. Каждый персонаж по умолчанию считается находящимся в уникальной саундгруппе.

2. Для того, чтобы объединить нескольких персонажей в единую саундгруппу, необходимо в секции логики прописать soundgroup = <текстовая строка>

Звуковые группы должны быть уникальными в пределах уровня, а еще лучше в пределах всей игры. Для этого указывайте в звуковой группе идентификатор уровня и сценки, например:

```
soundgroup = bar_dolg_kampfire1
```

3. Объединять в звуковые группы необходимо персонажей сидящих в кемпе и идущих в патрулях. А также при других схожих ситуациях.

4. Дабы избежать ошибок при обижании, наподобие той, которая сейчас проявляется в лагере на эксейпе, необходимо чтобы все НПС, логически относящиеся к одному лагерю имели одинаковый team, squad, group

**Пример:**

```
[kamp@esc_bridge_post1]  
center_point = kamp_point  
soundgroup = esc_bridge_soldiers
```