Гулаги(Динамические LTX)

Материал из S.T.A.L.K.E.R. Inside Wiki.

Перейти к: навигация, поиск

При создании гулага, не многие знают, что практически всю настройку логики гулагов, можно производить в файле «script». Такой способ построения достигается за счёт динамических ltx. Попробую описать, каким образом это достигается. Я думаю, многие заметили в файлах с гулагами такую функцию:

--'------------------------------------------------------------------------

--' Dynamic ltx

--'------------------------------------------------------------------------

function load\_ltx(gname, type)

return nil

end

Это именно та функция, которая и отвечает за динамические ltx. В данном случае, она отключена. Разберём простейший способ построения динамических ltx.

Создаём стандартным способом «smart\_terrain» в файле «all.spawn» и гулаг в файле «gulag\_уровень.script». Назначим ему имя, например «kakashki». Задаём ему несколько заданий. Например, зададим 3 кампа и двух уолкеров:

if type == "kakashki" then

for i = 1, 3 do

t = { section = "logic@kakashki\_kamp",

idle = 0,

prior = 8-i, state = {0,1},

in\_rest = "", out\_rest = ""

}

table.insert(sj, t)

end

for i = 1, 2 do

t = { section = "logic@kakashki\_walker\_"..i,

idle = 0,

prior = 9-i, state = {0,1},

in\_rest = "", out\_rest = ""

}

table.insert(sj, t)

end

end

Так как мы собираемся использовать динамические ltx, то дальнейшая настройка заданий будет производиться не в конфигурационных файлах, а непосредственно в этом же файле. То есть в упомянутой выше функции «load\_ltx». Для этого находим нужную функцию «load\_ltx», и в ней создаём динамический ltx и прописываем требуемые схемы логики:

--'------------------------------------------------------------------------

--' Dynamic ltx

--'------------------------------------------------------------------------

function load\_ltx(gname, type)

if type == "kakashki" then

local ltx = "[logic@kakashki\_kamp]\n"..

"active = kamp@kakashki\n"..

"[kamp@kakashki]\n"..

"center\_point = kamp\n"..

"[logic@kakashki\_walker\_1]\n"..

"active = walker@kakashki\_1\n"..

"[walker@kakashki\_1]\n"..

"path\_walk = walk\_1\n"..

"path\_look = look\_1\n"..

"[logic@kakashki\_walker\_2]\n"..

"active = walker@kakashki\_2\n"..

"[walker@kakashki\_2]\n"..

"path\_walk = walk\_2\n"..

"path\_look = look\_2\n"

return ltx

end

return nil

end

Где, кавычки обязательны, {\n} назначает новую строку в динамическом ltx, и {..} ставится для привязки строк между собой. В последней строке схем, двоеточие ставить не нужно, так как привязывать больше нечего. Переменная «ltx» может иметь любое другое имя.

Теперь создаём вэйпоинты путей, в файле «all.spawn», и можно смело идти проверять.

Одно из преимуществ динамических ltx - это пакетная настройка логики. Может, я не так выразился, но по ходу действий, я думаю вам станет ясно, что я имел ввиду. Например, нам нужно создать 8 уолкеров, с одинаковыми настройками логики, но при этом, разными поинтами путей. В статических ltx нам понадобится создавать 8 секций [logic]. Тоесть для каждого уолкера, своя секция логики. В динамических ltx, данную проблему можно решить функционально.

Делаем это так:

Создаём 8 уолкеров в функции "load\_job". Обязательно с помощью счётчика "for-do":

for i = 1, 8 do

t = { section = "logic@kakashki\_walker\_"..i,

idle = 0,

prior = 9-i, state = {0,1},

in\_rest = "", out\_rest = ""

}

table.insert(sj, t)

end

Кто ещё не знает, как это работает, то поясню. Создаётся переменная i, или любое другое имя. Назначается ей два, или три значения, (через запятую). Где:

первое значение = какое число задаётся при первом цикле счёта;

второе значение = при достижении или переполнении какого числа, остановить счёт;

третье значение = какое значение прибавлять, при каждом цикле счёта.

Если третье значение не задаётся, то при каждом цикле счёта, прибавляется 1.

Переменной "section" задаётся "имя секции"+переменная i. В итоге, при каждом последующем цикле счёта, к имени секции будет приписываться в конце значение переменной i. Тоесть, "logic@kakashki\_walker\_1"; "logic@kakashki\_walker\_2"; "logic@kakashki\_walker\_3" и т.д.

В нашем случае, переменной "prior" задано значение 9-i, что делать не обязательно. Здесь при каждом цикле счёта, вычитается значение переменной i от числа 9. В итоге, при каждом последующем цикле, переменной "prior" задаётся значение на 1 меньше предыдущего.

Итак, уолкеров создали. Теперь нам нужно задать им всем логику действий. Создаём переменную ltx, которую мы будем использовать для динамических ltx. Для этого, подымаемся на самый верх файла, где прописано создание переменной t, и ниже прописываем:

local ltx = ""

Теперь, в функции "load\_ltx" создаём ltx файл для нашего гулага:

function load\_ltx(gname, type)

if type == "kakashki" then

return ltx

end

return nil

end

Данным способом, мы возвращаем поиск секций логики, назад в функцию "load\_job".

Теперь возвращаемся в функцию "load\_job", и в секции нашего гулага, внутри счётчика "for-do" прописываем логику уолкерам. Выглядеть это будет так:

for i = 1, 8 do

t = { section = "logic@kakashki\_walker\_"..i,

idle = 0,

prior = 9-i, state = {0,1},

in\_rest = "", out\_rest = ""

}

table.insert(sj, t)

ltx = ltx.. "[logic@kakashki\_walker\_"..i.."]\n"..

"active = walker@kakashki\_"..i.."\n"..

"[walker@kakashki\_"..i.."]\n"..

"path\_walk = walk\_"..i.."\n"..

"path\_look = look\_"..i.."\n"

end

Где, так же как и в случае с переменной "section", приписывается в конец имён секций и путей, значение переменной i. В итоге получаем такие имена путей:

walk\_1, look\_1; walk\_2, look\_2; walk\_3, look\_3; и т.д.

Заметим, как мы прописали переменную ltx. Таким способом, мы при каждом цикле счёта, новую секцию логики пришиваем к секциям, пришитым к динамическому ltx при предыдущих циклах. Если прописать просто:

ltx = логика

То при каждом цикле счёта новая секция будет накладываться на предыдущую, что приведёт к нехорошим последствиям.

Ну и создаём соответствующие секции поинтов путей в файле all.spawn. И всё будет работать как часы.

Не забываем, при создании секций поинтов, к именам путей пришить имя смарт террэйна.

Продолжение следует.

Автор: Singapur22

Вот IG-2007 на оф.форуме осенью прошлого года написал такой тутор по гулагам. Здесь показана работа в гулаге для сталкера. Днем он гуляет (walker), а ночью сидит у костра (kamp).

text

1) Пропишите в all.spawn своему смарту такую custom\_data:

custom\_data = <<END

[smart\_terrain]

type = esc\_new\_lager

capacity = 1

END

2) Пропишите в all.spawn два пути: один из нескольких точек для схемы walker (esc\_new\_lager\_npc1\_walk), другой из одной точки (центр кампа) для kamp (esc\_new\_lager\_npc1\_kamp)

3) Откройте файл config\misc\gulag\_escape.ltx и добавьте в самый конец работу для своего сталкера:

[logic@esc\_new\_lager\_npc1]

active = walker@esc\_new\_lager\_npc1

[walker@esc\_new\_lager\_npc1]

path\_walk = npc1\_walk

on\_info = {!is\_day} kamp@esc\_new\_lager\_npc1

[kamp@esc\_new\_lager\_npc1]

center\_point = npc1\_kamp

on\_info = {=is\_day} walker@esc\_new\_lager\_npc1

4) Откройте файл gulag\_escape.script и добавьте в него:

4.1) в функцию load\_job

if type == "esc\_new\_lager" then

t = { section = "logic@esc\_new\_lager\_npc1",

idle = 0,

prior = 5, state = {0},

online = false,

in\_rest = "", out\_rest = ""

}

table.insert(sj, t)

end

4.2) в функцию load\_states

if type == "esc\_new\_lager" then

return function (gulag)

return 0

end

end

4.3) в функцию checkStalker

if gulag\_type == "esc\_new\_lager" then

return npc\_community == "stalker"

end

Можно. Сначала нужно исключить возможность попадания в гулаг. Для этого в логике пишется секция :

[smart\_terrains]

none = true

Далее логика пишется таким образом :

[logic]

active = kamp

[kamp]

center\_point = \*\*\* (\*\*\*-это имя точки, вокруг которой НПС садятся. Располагать её можно где угодно, не только у костра. Название этой точки должно быть уникальным, т.к. она прописывается в way\_"уровень".ltх)

radius = \* (\*-радиус в метрах от точки center\_point, в пределах которого НПС будут садится. Можно не указывать. По умолчанию 2 метра)

path\_walk = \*\*\*\_task (Можно не прописывать, если точка center\_point находится не в центре костра. \*\*\* - то же название, что и указано в center\_point, с добавлением \_task. Эта точка тоже прописывается в way\_"уровень".ltх, но координаты указываются на некотором расстоянии от center\_point. Эта строка нужна для того, чтобы при переходе из оффлайн НПС не попадали в костер и не получали хит, а выходили в этой точке)