**Скрипты в игре «Сталкер»**

**Документация для гейм-дизайнеров и скриптописателей**

1. В игре используется скриптовый язык Lua, поэтому необходимо воспользоваться его документацией для дальнейшего прочтения этого документа ([S:\GameData\Scripts\manual.html](file:///D:\SHB-Engine\Stalker%20Script%20Manual\manual.html))
2. Из игры в Lua вынесены следующие классы и глобальные функции:
   1. Базовые функции на C++
      1. log(<string\_to\_print>)
         1. выводит в лог строку
      2. flush()
         1. сохраняет лог на диск, работает очень медленно, в Release версии вообще отключена
      3. device()
         1. получить экземпляр класса render\_device (описание смотрите ниже)
      4. system\_ini() – возвращает указатель ini\_file на system.ltx
      5. alife\_simulator \*alife() – возвращает указатель на симулятор
      6. game.time()
         1. получить игровое время в миллисекундах
      7. level.object(<object\_name>)
         1. получить экземпляр класса game\_object по имени объекта
      8. level.actor()
         1. получить экземпляр класса game\_object актёра
      9. level.get\_weather()
         1. возвращает строку – имя текущей погоды
      10. level.set\_weather(<weather\_name>)
          1. устанавливает погоду
      11. level.set\_time\_factor(<time\_factor>)
          1. устанавливает game time factor
      12. level.get\_time\_factor()
          1. возвращает текущий game time factor
      13. level.cover\_in\_direction(<vertex\_id>,<direction>)
          1. возвращает прикрытость в заданном узле в заданном направлении
      14. level.vertex\_in\_direction(<vertex\_id>,<direction>,<max\_distance>)
          1. возвращает ноду в заданном напрвалении
      15. level.rain\_factor()
          1. возвращает, насколько идёт дождь (0..1)
      16. level.patrol\_path\_exists(<patrol\_path\_name>)
          1. возвращает, существует ли патрульный путь.
      17. level.vertex\_position(<vertex\_id>) – возвращает центр AI-ноды
   2. Базовые функции на Lua
      1. printf
         1. форматированный вывод данных, аналог printf в C++ (с ограничениями)
      2. wait()
         1. ждать до следующего обновления скрипта
      3. wait(<time\_to\_wait>)
         1. ждать заданное в миллисекундах время до следующего обновления скрипта
      4. wait\_game(<game\_time\_to\_wait>)
         1. ждать заданное в миллисекундах игровое время до следующего обновления скрипта
      5. action(<object>,<action1>,…)
         1. Добавить объекту object в очередь заданий новое с переданными под-action-ами (смотрите примеры)
   3. Классы C++
      1. vector
         1. класс трёхмерный вектор, имеет следующие члены, методы:

x,y,z – компоненты вектора

set – установить значения компонент вектора по заданным значениям (три числа, вектор)

add – добавить вектор

sub – отнять вектор

mul – умножить покомпонентно на вектор

div – поделить покомпонентно на вектор

… - есть ещё штук 30 других, но вы вряд ли будете ими пользоваться

set,

set,

add,

add,

add,

add,

sub,

sub,

sub,

sub,

mul,

mul,

mul,

mul,

div,

div,

div,

div,

invert,

invert,

min,

min,

max,

max,

abs,

similar,

set\_length,

align,

squeeze,

clamp,

clamp,

inertion,

average,

average,

lerp,

mad,

mad,

mad,

mad,

square\_magnitude,

magnitude,

magnitude,

normalize,

normalize,

normalize\_safe,

normalize\_safe,

dotproduct,

crossproduct,

distance\_to\_xz,

distance\_to\_sqr,

distance\_to,

from\_bary,

mknormal\_non\_normalized,

mknormal,

setHP,

getHP,

reflect,

slide,

generate\_orthonormal\_basis,

* + 1. matrix
       1. класс четырёхмерная матрица, имеет следующие члены, методы:

i,j,k,c – трёхмерные векторы

\_14\_,\_24\_,\_34\_,\_44\_ - числа

set – установить значения компонент вектора по заданным значениям (три вектора, матрица)

add – добавить матрицу

sub – отнять матрицу

mul – перемножить (матрицу на число, вектор, матрицу)

mul\_A – умножить на матрицу слева

mul\_B – умножить на матрицу справа

(умножение матриц не обладает коммутативностью, т.е. A\*B!= B\*A)

div – поделить покомпонентно на число

… - есть ещё штук 30 других, но вы вряд ли будете ими пользоваться

set

set

identity

mul

mul

mul

mul\_43

mulA

mulB

mulA\_43

mulB\_43

div

div

invert

invert

transpose

transpose,

translate,

translate,

translate\_over

translate\_over

translate\_add

scale

scale

rotateX

rotateY

rotateZ

rotation

rotation

mapXYZ

mapXZY

mapYXZ

mapYZX

mapZXY

mapZYX

mirrorX

mirrorX\_over

mirrorX\_add

mirrorY

mirrorY\_over

mirrorY\_add

mirrorZ

mirrorZ\_over

mirrorZ\_add

build\_projection

build\_projection\_HAT

build\_projection\_ortho

build\_camera

build\_camera\_dir

inertion

transform\_tiny

transform\_tiny

transform\_dir

transform\_dir

transform,

transform,

setHPB,

setXYZ,

setXYZ,

setXYZi,

setXYZi,

getHPB,

getHPB,

getXYZ,

getXYZ,

getXYZi,

getXYZi

* + 1. render\_device
       1. класс render\_device, имеет следующие члены:

width – ширина разрешения экрана в пикселях

height – высота разрешения экрана в пикселях

time\_delta – интервал между обновлениями устройства

time\_global – время клиентское

cam\_pos –позиция камеры

cam\_dir – направление камеры

cam\_top - \_j в матрице

cam\_right - \_i в матрице

view – матрица камеры

projection – матрица проекции

full\_transform – матрица преобразования

fov - fov

aspect\_ratio – aspect ratio

* + 1. sound\_object
       1. sound\_params

члены класса:

position - позиция

volume - громкость

frequency - частота

min\_distance – минимальное расстояние слышимости звука

max\_distance – максимальное расстояние слышимости звука

члены класса:

frequency

min\_distance

max\_distance

volume

конструкторы и методы

конструктор без параметров – создаёт объект с default-настройками

конструктор с параметром <звуковой файл>

конструктор с параметром <звуковой файл, тип звука>

get\_position() – получить позицию

set\_position(<new\_position>) – установить позицию

play(<object>,<delay\_in\_seconds>,<flags>) – играть с указанными флагами :

sound\_object.looped,

sound\_object.s2d,

sound\_object.s3d

play\_at\_pos(<object><position>,<delay\_in\_seconds>,<flags>) – играть в позиции с указанными флагами

play\_clone(<object>,<delay\_in\_seconds>,<flags>) – играть клон звука (доступа к нему нет), флаги не могут содержать зацикленность

play\_at\_pos\_ clone (<object><position>,<delay\_in\_seconds>,<flags>) – играть в позиции клон звука (доступа к нему нет), флаги не могут содержать зацикленность

stop() – остановить звук

playing() – играет звук или уже нет

length() – длина звука в миллисекундах

* + 1. hit

члены класса:

power - сила

direction - напрваление

draftsman – кто наносит хит

impulse – физический импульс

type – тип хита, бывает следующий

hit.burn

hit.shock

hit.strike

hit.wound

hit.radiation

hit.telepatic

hit.chemical\_burn

hit.explosion

hit.fire\_wound

конструктор без параметров – создаёт объект с default-настройками.

методы :

bone (<bone\_name>) – задаёт имя кости, куда наносится хит

* + 1. move
       1. patrol

конструкторы:

без параметров – создаёт объект с default-настройками

<имя пути>,

<тип старта>,

<тип финиша>,

<если путь имеет ветвления, то выбирать ли их случайно>

тип старта:

patrol.start

patrol.finish

patrol.nearest

тип финиша:

patrol.stop

patrol.continue

первый параметр – обязательный, остальные – нет,

по умолчанию

тип старта = patrol.nearest,

тип финиша = patrol.continue

случайность = true

методы:

count() –

возвращает количество waypoint-ов в пути

point(<индекс>) –

возвращает вектор, соответствующий waypoint-у с указанным индексом

level\_vertex\_id(<индекс>) –

возвращает вектор, соответствующий waypoint-у с указанным индексом

index(<имя\_waypoint-а>) -

возвращает индекс waypoint-а с указанным именем

nearest(<вектор>)

возвращает индекс waypoint-а, ближайшего к заданной точке

flag(<индекс>,<бит>) –

возвращает TRUE или FALSE для указанного бита указанного waypoint-а патрульного пути

конструкторы

1. без параметров – создаёт объект с default-настройками
2. <input\_key>, <speed>

Этот конструктор предназначен только для машин.

input\_key бывают следующими:

move.none – никаких команд

move.fwd - вперёд

move.back - назад

move.left - влево

move.right - вправо

move.up – вверх на одну передачу

move.down – вниз на одну передачу

move.on – включить зажигание

move.off – выключить зажигание

speed – задаёт ограничение скорости, если не параметр не задан, то он равен нулю, тогда скорость не ограничивается

Для управления машиной нужно использовать только этот конструктор

1. <body\_state>,<movement\_type>,<path\_type>,<object>,

<speed>

body\_state бывает следующим:

move.crouch - сидя

move.standing - стоя

movement\_type бывает следующим:

move.stand - стоять

move.walk - идти

move.run - бежать

path\_type бывает следующим:

move.line – прямая линия

move.dodge – искривлённая линия

move.criteria – по критерию (пока не работает)

move.curve – сглаженный путь (пока не работает)

move.curve\_criteria – сглаженный путь по критерию (пока не работает)

object – объект, за которым нужно следовать

speed – скорость объекта, если не задана, объект сам назначает свою скорость, в зависимости от переданных параметров

1. <body\_state>,<movement\_type>,<path\_type>,<patrol>,

<speed>

patrol – экземпляр класса патрульного пути, который описан выше

1. <body\_state>,<movement\_type>,<path\_type>,<position>,

<speed>

position – позиция, куда нужно идти

1. <position>,<speed>

Если вы хотите направить объект, куда угодно не по нодам, то нужно указать позицию и скорость передвижения

Кроме того, можно менять все свойства у существующего экземпляра класса с помощью следующих методов:

body

move

path

object

patrol

position

input

* + 1. look

Конструкторы:

1. без параметров – создаёт объект с default-настройками
2. <look\_type>

look\_type бывает следующим:

look.path\_dir – смотреть по пути

look.search – смотреть по сторонам так, чтобы не менять направление тела, которое ориентировано по пути

look.danger – смотреть по сторонам куда хочу, даже если придётся менять направление тела

look.point – смотреть в точку только головой

look.fire\_point – смотреть в точку головой и автоматом

look.direction – смотреть в каком-то направлении

1. <look\_type>,<direction>

direction – вектор, куда смотреть

1. <look\_type>,<object>

object – объект, на который смотреть

Кроме того, можно менять все свойства у существующего экземпляра класса с помощью следующих методов:

object

direct

type

* + 1. anim

Конструкторы:

1. без параметров – создаёт объект с default-настройками
2. <название\_анимации>
3. <mental\_state>

mental\_state бывает следующим:

anim.free – спокойное состояние

anim.danger – возбуждённое состояние

anim.asleep – пока не реализовано

anim.zombied – пока не реализовано

Если задана анимация, то персонаж её отыгрывает, если же задан тип анимаций, то он сам следит за тем, какие анимации играть

Кроме того, можно менять все свойства у существующего экземпляра класса с помощью следующих методов:

anim

type

* + 1. sound

Конструкторы

* + - 1. без параметров – создаёт объект с default-настройками
      2. <sound\_name>, <bone\_name>, <position\_offset>, <angle\_offset>, <looped>, <snd\_type>

sound\_name – имя звука

bone\_name – имя кости

position\_offset – вектор, смещение относительно позиции кости

angle\_offset – вектор, угловое смещение относительно углов кости

looped – зацикленный звук

snd\_type – тип звука

первые два параметра обязательны, остальные по умолчанию равны нулю

* + - 1. <sound\_name>, <position>, <angle\_offset>, <looped>, <snd\_type>

position – место, где играть звук

Кроме того, можно менять все свойства у существующего экземпляра класса с помощью следующих методов:

set\_sound

set\_sound(<sound\_object>)

set\_sound\_type(<snd\_type>)

set\_bone

set\_position

set\_angles

* + 1. particle
       1. particle\_param

Конструктор <position\_offset>,<angle\_offset>,<speed>

speed – скорость частиц, все параметры необязательны

Конструкторы

* + - 1. без параметров – создаёт объект с default-настройками
      2. <particle\_name>, <bone\_name>, <particle\_params>, <looped>

particle \_name – имя системы частиц

bone\_name – имя кости

particle\_params – экземпляр класса particle\_param

looped – зацикленный партикл

первые два параметра обязательны, остальные по умолчанию равны нулю

* + - 1. < particle \_name>, <position>, <angle\_offset>, <looped>

position – место, где играть партиклы

Кроме того, можно менять все свойства у существующего экземпляра класса с помощью следующих методов:

set\_particle

set\_bone

set\_position

set\_angles

set\_velocity

* + 1. object

Конструкторы:

* + - 1. без параметров – создаёт объект с default-настройками
      2. <game\_object>, <action\_type>, <queue\_size>

game\_object – объект, с которым нужно что-то делать

action\_type бывает следующим:

object.idle – просто держать

object.fire1 – стрелять1

object.fire2 – стрелять2

object.reload – перезарядить

object.activate – активизировать

object.deactivate - деактивизировать

object.use – использовать

object.take – поднять

object.drop – бросить

queue\_size – размер очереди при стрельбе

* + - 1. <bone\_name>, <action\_type>

Этот конструктор только для указаний действий для машины

bone\_name – имя кости (например, для того, чтобы открытьт/закрыть дверь в машине (activate\deactivate\use(если открыта – закрыть, если закрыта - открыть)))

* + 1. cond

Конструкторы:

* + - 1. без параметров – создаёт объект с default-настройками
      2. <condition\_type>,<time>

condition\_type бывает следующим:

cond.move\_end – окончание движения

cond.look\_end – окончание изменения ориентации в пространстве

cond.anim\_end – окончание анимации

cond.sound\_end – окончание звука

cond.object\_end – окончание выполнения действия

cond.time\_end – окончания времени

condition\_type может быть составным, т.е. cond.move\_and + cond.time\_end, что будет означать, что закончить действие если закончено движение или время истекло

Если задано условие cond.time\_end, то нужно указать, сколько именно времени нужно ждать

* + 1. entity\_action

Конструкторы:

* + - 1. без параметров – создаёт объект с default-настройками

Методы:

set\_action(…) – параметром может быть экземпляр любого из классов “…\_action”

move(),

look(),

anim(),

sound(),

object(),

time(),

all() – функции проверки выполнения соответствующих условий

* + 1. rotation – члены класса :

yaw,

pitch - углы по двум осям

* + 1. object\_params – члены класса :

level\_vertex – нода АИ-шной карты

position – позиция

orientation – экземпляр класса rotation

* + 1. memory\_object – члены класса :

game\_time – игровое время

level\_time – реальное время

update\_count – количество обновлений объекта памяти

* + 1. entity\_memory\_object – порождён от класса memory\_object

члены класса :

object\_info – информация об объекте, экзмепляр класса object\_params

self\_info – информация о себе, экзмепляр класса object\_params

object() – получить game\_object объекта

* + 1. game\_memory\_object – порождён от класса memory\_object

члены класса :

object\_info – информация об объекте, экзмепляр класса object\_params

self\_info – информация о себе, экзмепляр класса object\_params

object() – получить game\_object объекта

* + 1. hit\_memory\_object – порождён от класса entity\_memory\_object

члены класса :

direction – направление, откуда пришёл хит

bone\_index – индекс кости, в которую попал хит

amount – сколько здоровья отнял

* + 1. visible\_memory\_object – порождён от класса game\_memory\_object

члены класса :

visible – видим ли объект сейчас

* + 1. memory\_info - порождён от класса visible\_memory\_object

члены класса :

visual\_info – находится ли объект в визуальной памяти

sound\_info – находится ли объект в звуковой памяти

hit\_info – находится ли объект в хитовой ☺ памяти

* + 1. sound\_memory\_object – порождён от класса visible\_memory\_object

члены класса:

power – сила звука

методы:

type() – получить тип звука, он может быть следующим :

snd\_type.no\_sound - 0x00000000

snd\_type.weapon - 0x80000000

snd\_type.item - 0x40000000

snd\_type.monster - 0x20000000

snd\_type.anomaly - 0x10000000

snd\_type.world - 0x08000000

snd\_type.pick\_up - 0x04000000

snd\_type.drop - 0x02000000

snd\_type.hide - 0x01000000

snd\_type.take - 0x00800000

snd\_type.use - 0x00400000

snd\_type.shoot - 0x00200000

snd\_type.empty - 0x00100000

snd\_type.bullet\_hit - 0x00080000

snd\_type.reload - 0x00040000

snd\_type.die - 0x00020000

snd\_type.injure - 0x00010000

snd\_type.step - 0x00008000

snd\_type.talk - 0x00004000

snd\_type.attack - 0x00002000

snd\_type.eat - 0x00001000

snd\_type.idle - 0x00000800

snd\_type.object\_break - 0x00000400

snd\_type.object\_collide - 0x00000200

snd\_type.object\_explode - 0x00000100

snd\_type.ambient - 0x00000080

-- далее идут комбинации однобитовых масок

snd\_type.item\_pick\_up - 0x44000000

snd\_type.item\_drop - 0x42000000

snd\_type.item\_hide - 0x41000000

snd\_type.item\_take - 0x40800000

snd\_type.item\_use - 0x40400000

snd\_type.weapon\_shoot - 0x80200000

snd\_type.weapon\_empty - 0x80100000

snd\_type.weapon\_bullet\_hit - 0x80080000

snd\_type.weapon\_reload - 0x80040000

snd\_type.monster\_die - 0x20020000

snd\_type.monster\_injure - 0x20010000

snd\_type.monster\_step - 0x20008000

snd\_type.monster\_talk - 0x20004000

snd\_type.monster\_attack - 0x20002000

snd\_type.monster\_eat - 0x20001000

snd\_type.anomaly\_idle - 0x10000800

snd\_type.world\_object\_break - 0x08000400

snd\_type.world\_object\_collide - 0x08000200

snd\_type.world\_object\_explode - 0x08000100

snd\_type.world\_object\_ambient - 0x08000080

каждый тип звука является битовой маской, т.е. вы можете узнать по звуку, например, является ли он звуком оружия или выстрелом.

Для битовых операций вынесены следующие функции :

bit\_and(<number>,<number>)

bit\_or(<number>,<number>)

bit\_xor(<number>,<number>)

bit\_not(<number>)

Для получения объектов из классов memory\_object вынесено пока только 2 прототипа функции :

get\_memory\_object(<hit\_memory\_object>)

get\_memory\_object(<snd\_type.object>)

* + 1. not\_yet\_visible\_object
       1. член класса value – значение, насколько объект видим
       2. object() – получить объект, который собирается стать или уже стал видимым
    2. abstract\_evaluator – пока пустой
    3. cover\_point
       1. position() – получить позицию
       2. level\_vertex\_id() – получить индекс ноды
    4. game\_object

Члены класса:

visible – видимость

enabled – доступность

health – здоровье

power – сила

satiety – сытость

radiation – радиация

circumspection – осторожность

morale – мораль

Конструкторы:

* + - 1. без параметров – создаёт объект с default-настройками

Методы:

position() – получить позицию

class\_id() – идентификатор класса

id() – идентификатор объекта

section() – секция в system.ltx

name() - имя

parent() – кому принадлежит

mass() - масса

cost() - стоимость

death\_time() – время смерти, если помер

armor() - броня

max\_health() – максимальное здоровье

accuracy() - точность

alive() – живой или неживой

team() – команда

squad() – отряд

group() – группа

kill(<who>) – убить себя

hit(hit) – нанести хит

fov() - fov

range() – максимальная видимость

relation(game\_object) – отношение в объекту (friend,neutral,enemy,dummy)

script(<true/false><script\_name>) – установить в скриптовый режим

get\_script() – получить скриптовый режим

get\_script\_name() – получить имя скрипта

see(<game\_object>) – проверить, видит ли заданный объект

see(“section\_name”) – проверить, видит ли заданный класс, в кавычках – имя секции из system.ltx

who\_hit\_name – возвращает имя объекта, нанесшего последний хит

who\_hit\_section\_name – возвращает имя секции того типа объектов, нанесших последний хит

use(<game\_object>) – использовать что-то (пока не работает)

rank() – получить ранг сталкера

get\_ammo() – получить количество патронов

command(<entity\_action>) – отдать команду

action() – получить текущую команду

object\_count() – получить количество объектов в рюкзаке

object(<object\_name>) – получить объект из рюкзака по секции

object(<object\_index>) – получить объект из рюкзака по индексу

active\_item() – получить текущий активный объект

set\_callback(<callback>,<true/false>) – установить callback на вход/выход из зоны

clear\_callback(<true/false>) – снять callback на вход/выход из зоны

set\_callback(<callback>,<action\_type>) – установить callback на указанный action.

<action\_type> :

movement,

watch,

animation,

sound,

particle,

object

clear\_callback(< action\_type >) – снять callback с указанного action-a

* + 1. physics\_shell

члены класса: нет

конструкторы: нет (получить у game\_object)

методы:

apply\_force(<number>,<number>,<number>)

приложить силу ко всем элементам. Аргументы – вектор в глобальных координатах

get\_element\_by\_bone\_name(<string>)

get\_element\_by\_bone\_id (<number>)

get\_element\_by\_order (<number>) –

получить элемент

get\_elements\_number()

получить количество элементов

get\_joint\_by\_bone\_name (<string>)

get\_joint\_by\_bone\_id (<number>)

get\_joint\_by\_order (<number>) –

получить джоинт

get\_joints\_number()

получить количество джоинтов

* + 1. physics\_element

члены класса: нет

конструкторы: нет (получить у physics\_shell)

методы:

apply\_force(<number>,<number>,<number>)

* + 1. physics\_joint

get\_bone\_id()

get\_first\_element()

get\_first\_element()

set\_anchor\_global(<number>,<number>,<number>)

set\_anchor\_vs\_first\_element(<number>,<number>,<number>) set\_anchor\_vs\_second\_element(<number>,<number>,<number>)

set\_axis\_spring\_dumping\_factors((<number>,<number>,<number>)

Задать множители для упругости и дампинга (как бы вязкость), лимитов установленных на оси. Аргументы: упругость, дампинг, номер оси.

set\_joint\_spring\_dumping\_factors(((<number>,<number>)

set\_axis\_dir\_global (<number>,<number>,<number>>,<number>)

set\_axis\_dir\_vs\_first\_element (<number>,<number>,<number>>,<number>)

set\_axis\_dir\_vs\_second\_element(<number>,<number>,<number>,<number>)

Агументы: вектор в соответствующих координатах,номер оси

set\_limits(<number>,<number>,<number>)

Аргументы: нижний предел, верхний предел, номер оси.

Пределы задаются в радианах

нижний от 0 до –Пи, верхний от 0 до +Пи.

set\_max\_force\_and\_velocity(<number>,<number>,<number>)

устанавливает скорость и максимальную силу, необходимую для достижения этой скорости. Аргументы:

сила, скорость, номер оси джоинта. Скорость радианы/секунду. Сила – вращательный момент в Ньютон\*метр. [Ньютон]=[кг]\*[м]/[с]^2

get\_max\_force\_and\_velocity()

get\_axis\_angle()

get\_limits()

get\_axis\_dir()

get\_anchor()

String Table

get\_by\_id - получение записи таблицы по строковому id

get\_by\_index– получение записи таблицы по числовому индексу

index\_by\_id – получение числовому индексу по строковому id

Пример использования:

local st = string\_table()

text = st:get\_by\_id("inventory\_item\_use")

Usable Script Callback

Callback, который можно поставить на использование вещи актером

set\_use\_callback (<callback>)

clear\_use\_callback(<callback>)

set\_tip\_text(<text>) – установка строки-подсказки, которая будет показана при наведении на предмет курсором

set\_tip\_text\_default() – возвращение строки-подсказки по умолчанию

set\_nonscript\_usable(<true/false>) – разрешение/запрещение стандартных (нескриптовых) действий над объектом (взять объект в интерфейс, говорить с персонажем и т.д.)

PDA

ID сообщений которые можно посылать и получать функцией

send\_pda\_message(<pda\_msg\_id>, <who>)

trade\_pda\_msg

help\_pda\_msg

go\_away\_pda\_msg

info\_pda\_msg

accept\_pda\_msg

decline\_pda\_msg

i\_leave\_pda\_msg

no\_pda\_msg

<true, false> need\_to\_answer\_pda - есть ли сообщения на которые нужно ответить актеру

answer\_pda() – ответить на сообщения

give\_info\_portion(<info\_index>) – получить информацию по индексу

give\_info\_portion\_via\_pda(<info\_index>, <who>) – получить информацию по индексу от указанного объекта

disable\_info\_portion(<info\_index>)

has\_info(<info\_index>)

dont\_has\_info(<info\_index>)

is\_talking ()

stop\_talk ()

enable\_talk ()

disable\_talk ()

is\_talk\_enabled ()

transfer\_item (<item>, <who>) – передача объекта из нашего инвентаря в чужой

transfer\_money (<amount>, <who>) – передача денег из нашего инвентаря в чужой

money () – количество денег

------------------------------------------------------------------

Character Info

set\_goodwill (<goodwill>, <who>) – установка благосклонности к кому-либо

goodwill (<who>) – текущая благосклонность

set\_relation (<relation>, <who>) – установка отношения

set\_start\_dialog(<dialog\_id>) – установка приветственного диалога NPC

get\_start\_dialog () – текучий диалог

restore\_default\_start\_dialog () – восстановление диалога, заданного в профиле персонажа

character\_name

character\_rank

set\_character\_rank(<int 0 to 100>)

character\_reputation()

set\_character\_reputation (<int 0 to 100>)

character\_community

set\_character\_community(<LPSCTR community>)

------------------------------------------------------------------

Actor

Во время разговора:

switch\_to\_trade() - переключение из режима диалога в режим торговли

switch\_to\_talk()- переключение из режима торговли в режим диалога

run\_talk\_dialog(to\_who) – принудительный запуск диалога с актера с NPC, to\_who – NPC персонаж

actor\_sleep (hours, minutes) – принудительный сон актера hours часов и minutes минут

------------------------------------------------------------------

Script Task

Состояние подцели

<objective\_state> = {task.fail, task.in\_progress, task.completed, task.dummy}

Для актера можно с сюжетными скриптами

<objective\_state> get\_task\_state (<LPCSTR task\_id>, <objective\_num>)

если такого задания у актера нет, то вернется task.dummy

set\_task\_state (<objective\_state>, <LPCSTR task\_id>, <objective\_num>)

если такого задания у актера нет, то выдаст ошибку

------------------------------------------------------------------

patrol() – получить текущий патрульный путь

set\_trade\_callback(<callback>) – установить callback на продажу/покупку

clear\_trade\_callback() – снять callback с продажи/покупки

get\_ammo\_in\_magazine() – вернуть количество патронов в магазине

get\_ammo\_total() – вернуть общее количество патронов в рюкзаке, спрашивается у автомата, который уже находится у сталкера или актёра

set\_queue\_size(<queue\_size>) – установить размер очереди при стрельбе из автоматического оружия

best\_hit() – возвращает hit\_memory\_object

best\_sound() – возвращает snd\_type.object

best\_enemy() – возвращает объект

best\_item() – возвращает объект

action\_count() – возврвщает количество action-ов в очереди

action\_by\_index(<action\_index>) – получает по индексу action

set\_hit\_callback(<callback>) – устанваливает callback (функцию и/или метод экземпляра класса) на hit. Callback-функция/ метод экземпляра класса должна(ен) принимать следующие параметры :

hit\_callback(self, amount, local\_direction, who, bone\_index)

clear\_hit\_callback(<is\_member>) – снимает callback на hit, параметр указывает что снимать – функцию или член экземпляра класса

set\_hear\_callback(<callback>) – устанваливает callback (функцию и/или метод экземпляра класса) на hear. Callback-функция/ метод экземпляра класса должна(ен) принимать следующие параметры :

hear\_callback(self, who, sound\_type, sound\_position, sound\_power)

clear\_hear\_callback(<is\_member>) – снимает callback на hear, параметр указывает что снимать – функцию или член экземпляра класса

memory(<object>) – спросить у объекта, что он помнит об заданном объекте, возвращает экземпляр класса memory\_info

best\_weapon() – получить от объекта его лучшее оружие (только для сталкеров)

explode() – взорвать объект (если он является гранатой)

bind\_object(<game\_object>) – прибиндить объект (только для сталкеров)

enum global\_path\_type {

game\_path,

level\_path,

enemy\_search – not implemented,

patrol\_path,

no\_path

}

< motivation\_action\_manager > motivation\_action\_manager()

void set\_body\_state(<body\_state>)

void set\_movement\_type(<movement\_type>)

void set\_mental\_state(<mental\_state>)

void set\_path\_type(<global\_path\_type >)

void set\_detail\_path\_type(<path\_type>)

void set\_desired\_position(fvector \* = 0)

void set\_desired\_direction(fvector \* = 0)

void set\_node\_evaluator(abstract\_evaluator \*=0)

void set\_path\_evaluator(abstract\_evaluator \*=0)

void set\_patrol\_path(<name>, <patrol\_start\_type>, <patrol\_route\_type>, bool random)

void add\_sound(<prefix>, <sound\_type>, <priority>, <mask>, <internal\_type>)

void set\_sound\_mask(<u32>)

void play\_sound (<id>,max\_start\_time,min\_start\_time,max\_stop\_time,min\_stop\_time)

void set\_item(<object.action\_type>,game\_object \*=0)

void set\_sight(<look\_type>)

u32 level\_vertex\_id()

void set\_dest\_level\_vertex\_id(<vertex\_id>)

void add\_animation(<animation\_name>,bool hand\_usage)

(если hand\_usage, то использовать только одну (правую) руку для вычисления направления оружия)

void clear\_animations()

int animation\_count()

ini\_file \*spawn\_ini() возвращает указатель на custom data

const cover\_point \*best\_cover (<self\_position>, <enemy\_position>, <radius>, <min\_enemy\_distance>, <max\_enemy\_distance> ) – возвращает лучшую точку приврытия от врага

const vector<CVisibleObject> &memory\_visible\_objects() const – объекты из памяти, которые видны или были когда-то видны. – итерировать как по таблице LUA.

for a in object: memory\_visible\_objects() do

….

end

const vector<CSoundObject> &memory\_sound\_objects() const – объекты из памяти, которые слышны или были когда-то слышны. – итерировать как по таблице LUA.

const vector<CHitObject> &memory\_hit\_objects() const – объекты из памяти, которые наносят или наносили повреждения. – итерировать как по таблице LUA.

const vector<CNotYetVisibleObject> &not\_yet\_visible\_objects() const – объекты из памяти, которые видны или собираются стать видны. – итерировать как по таблице LUA.

void enable\_memory\_object(memory\_object&, bool enable) – активизировать/деактивизировать объект из памяти

float visibility\_threshold() const – получить пороговое значение. Если not\_yet\_visible\_object::value имеет значение >= этого, то объект считается видимым.

vector bone\_position(<bone\_name>) const – возвращает позицию кости.

void add\_restrictions(<out\_restrictions\_names>, <in\_restrictions\_names>)

void remove\_restrictions(<out\_restrictions\_names>, <in\_restrictions\_names>)

void remove\_all\_restrictions()

LPCSTR out\_restrictions() const

LPCSTR in\_restrictions() const

bool accessible(const Fvector &)

bool accessible(level\_vertex\_id)

u32,result accessible\_nearest(const Fvector &position, Fvector &result)

* + 1. effector
       1. duality

h,v – горизонтальная и вертикальная

* + - 1. color

r,g,b,a – red/gree/blue/alpha

* + - 1. noise

intensity,grain,color – интенсивность, гранулярность и цвет (описан выше)

* + - 1. effector\_params

blur, gray, dual, noise – блур, цветность, дуальность (описано выше), шум (описан выше)

Члены класса:

info – то, куда нужно сохранить изменённые в process() параметры, чтобы они были применены (это из-за того, что Lua не понимает конструкции non-const object reference)

Конструктор:

<effector\_type>,<time\_to\_play>

effector\_type – тип эффектора (должен быть уникальным)

time\_to\_play – сколько играть

Методы:

add() – добавить эффектор (менеджер камер сразу начнёт вызывать у него метод process)

remove() – удалить эффектор

process(<effector\_params>) – что-то делать с параметрами

* + 1. property\_storage

Конструктор: - без параметров

Методы:

void set\_property (<property\_id>,<bool>)

bool property (<property\_id>)

* + 1. world\_property

Конструктор:

(<property\_id>,<value>)

Методы:

property\_id condition ()

value value ()

* + 1. world\_state

Конструктор:

()

(<world\_state>)

Методы:

void add\_property (<world\_property>)

void remove\_property (<property\_id>)

void clear ()

bool includes (<world\_state>)

value property (<property\_id>)

Операторы:

<

==

* + 1. property\_evaluator

Члены класса:

object – game\_object

storage – property\_storage

Конструктор:

()

(game\_object)

Методы:

virtual void reinit ()

virtual value evaluate ()

* + 1. action\_base

Члены класса:

object – game\_object

storage – property\_storage

Конструктор:

()

(game\_object)

(game\_object,string)

Методы:

virtual void reinit (game\_object,clear\_all)

virtual void initalize ()

virtual void finalize ()

virtual void execute ()

void add\_precondition (<world\_property>)

void remove\_precondition (<property\_id>)

void add\_effect (<world\_property>)

void remove\_effect (<property\_id>)

void set\_weight (<edge\_value\_type> action\_weight)

* + 1. action\_planner

Члены класса:

object – game\_object

storage – property\_storage

Конструктор:

()

Методы:

virtual void reinit (game\_object,clear\_all)

virtual void update ()

void add\_action (<action\_id>,<action\_base>)

void remove\_action (<action\_id>)

<action\_base> action (<action\_id>)

void add\_evaluator (<property\_id>, <property\_evaluator>)

void remove\_evaluator (<property\_id>)

<property\_evaluator> evaluator (<property\_id>)

<action\_id> current\_action\_id ()

<action\_base> current\_action ()

bool initalized ()

void set\_goal\_world\_state (<world\_state>)

void clear ()

* + 1. planner\_action : action\_base, action\_planner

Конструктор:

()

Методы:

virtual void reinit (game\_object,clear\_all)

* + 1. motivation

Члены класса:

object – game\_object

Конструктор:

()

Методы:

virtual void reinit (<game\_object>)

virtual void load (<string>)

virtual void reload (<string>)

virtual float evaluate (<motivation\_id>)

* + 1. motivation\_action : motivation

Конструктор:

(<world\_state>)

Методы:

<world\_state> goal ()

* + 1. motivation\_manager

Конструктор:

()

Методы:

void add\_motivation (<motivation\_id>,<motivation>)

void remove\_motivation (<motivation\_id>)

void add\_connection (<motivation\_id>,<motivation\_id>)

void remove\_connection (<motivation\_id>,<motivation\_id>)

<motivation> motivation (<motivation\_id>)

<motivation> selected ()

<motivation\_id> selected\_id ()

void clear ()

virtual void reinit (<game\_object>)

virtual void load (<string>)

virtual void reload (<string>)

virtual void update (<u32>)

* + 1. motivation\_action\_manager : motivation\_manager, action\_planner

Конструктор:

()

Методы:

virtual void clear ()

void clear\_motivations ()

void clear\_actions ()

void clear ()

virtual void reinit (<game\_object>)

virtual void load (<string>)

virtual void reload (<string>)

virtual void update (<u32>)

* + 1. server\_object – пока пустой класс
    2. net\_packet – пока пустой класс
    3. object\_binder

Члены класса:

object – game\_object

Конструктор:

(<game\_object>)

Методы:

void clear ()

virtual void reinit (<game\_object>)

virtual void reload (<string>)

virtual void net\_spawn (<server\_object>)

virtual void net\_destroy ()

virtual void net\_import (<net\_packet>)

virtual void net\_export (<net\_packet>)

* + 1. stalker\_ids – пустой класс, нужен для id-шников motivations, properties, actions у сталкера:

motivation\_global

motivation\_alive

motivation\_dead

motivation\_solve\_zone\_puzzle

motivation\_script

property\_already\_dead

property\_dead

property\_puzzle\_solved

property\_alive

property\_alife

property\_items

property\_enemy

property\_see\_enemy

property\_item\_to\_kill

property\_found\_item\_to\_kill

property\_item\_can\_kill

property\_found\_ammo

property\_ready\_to\_kill

property\_kill\_distance

property\_anomaly

property\_inside\_anomaly

property\_script

action\_squad\_action

action\_death\_planner

action\_already\_dead

action\_dead

action\_puzzle\_solver

action\_free\_no\_alife

action\_free

action\_gather\_items

action\_combat\_planner

action\_get\_kill\_distance

action\_get\_enemy

action\_get\_enemy\_seen

action\_get\_item\_to\_kill

action\_find\_item\_to\_kill

action\_make\_item\_killing

action\_find\_ammo

action\_get\_ready\_to\_kill\_very\_aggressive

action\_kill\_enemy\_very\_aggressive

action\_get\_ready\_to\_kill\_ aggressive

action\_kill\_enemy\_ aggressive

action\_get\_ready\_to\_kill\_ moderate

action\_kill\_enemy\_moderate

action\_get\_ready\_to\_kill\_avoid

action\_kill\_enemy\_avoid

action\_retreat\_from\_enemy

action\_anomaly\_planner

action\_get\_out\_of\_anomaly

action\_detect\_anomaly

action\_script

sound\_die

sound\_injuring

sound\_humming

sound\_alarm

sound\_surrender

sound\_script

* + 1. abstract
       1. u16 id() const – возвращает уникальный идентификатор объекта
       2. u16 parent\_id() const – возвращает уникальный идентификатор владельца объекта
    2. alife\_object : abstract
       1. конструктор (<section\_name>)
       2. bool can\_switch\_online() const – может ли объект переходить в онлайн
       3. bool can\_switch\_offline() const – может ли объект переходить в оффлайн
       4. bool interactive() const – является ли объект интерактивным в симуляции
    3. alife\_dynamic\_object : alife\_object
       1. конструктор (<section\_name>)
    4. alife\_simulator
       1. alife\_dynamic\_object \*object(<object\_id>) – получить серверный объект по идентификатору
       2. alife\_dynamic\_object \*story\_object(<story\_object\_id>) – получить серверный объект по сюжетному идентификатору (Story ID в Level Editor)
       3. set\_switch\_online(<object\_id>,<new\_value>) – устанавливает, может ли объект переходить в онлайн
       4. set\_switch\_offline(<object\_id>,<new\_value>) – устанавливает, может ли объект переходить в оффлайн
       5. set\_interactive(<object\_id>,<new\_value>) – устанавливает, является ли объект интерактивным в симуляции
    5. clsid – пустой класс, нужен для идентификаторов классов

level

game

hud\_manager

custom\_target

rat\_group

flesh\_group

spawn\_group

event

graph\_point

actor

spectator

rat

flesh

chimera

dog\_red

soldier

stalker

zombie

idol

bloodsucker

boar

dog\_black

burer

pseudo\_gigant

controller

trader

crow

car

helicopter

art\_mercury\_ball

art\_black\_drops

art\_needles

art\_bast\_artefact

art\_gravi\_black

art\_dummy

art\_zuda

art\_thorn

art\_faded\_ball

art\_electric\_ball

art\_rusty\_hair

art\_galantine

art\_gravi

artefact

net\_target\_assault

net\_target\_cs\_base

net\_target\_cs\_cask

net\_target\_cs

wpn\_fn2000

wpn\_ak74

wpn\_lr300

wpn\_hpsa

wpn\_pm

wpn\_fort

wpn\_binocular

wpn\_shotgun

wpn\_svd

wpn\_svu

wpn\_rpg7

wpn\_val

wpn\_vintorez

wpn\_walther

wpn\_usp45

wpn\_groza

wpn\_knife

-----------------

wpn\_ammo

wpn\_ammo\_vog25

wpn\_ammo\_og7b

wpn\_ammo\_m209

-----------------

wpn\_scope

wpn\_silencer

wpn\_grenade\_launcher

obj\_bolt

obj\_medkit

obj\_antirad

obj\_food

obj\_bottle

obj\_explosive

obj\_document

obj\_attachable

equ\_scientific

equ\_stalker

equ\_military

equ\_exo

wpn\_grenade\_f1

wpn\_grenade\_rgd5

wpn\_grenade\_rpg7

wpn\_grenade\_fake

zone

zone\_mosquito\_bald

zone\_mincer

zone\_acid\_fog

zone\_galantine

zone\_radioactive

zone\_bfuzz

zone\_rusty\_hair

zone\_dead

level\_changer

script\_zone

team\_base\_zone

device\_detector\_simple

device\_torch

device\_pda

device\_art\_merger

projector

wpn\_mounted

hanging\_lamp

obj\_physic

script\_object

obj\_breakable

* + 1. класс token
       1. конструктор без параметров
       2. члены класса :

name - string

id – u32

* + 1. класс token\_list
       1. конструктор без параметров
       2. add(<name>,<id>)
       3. remove(<name>)
       4. clear()
       5. id(<name>)
       6. name(<id>)
    2. класс ini\_file :
       1. конструктор (<имя файла>) – без пути, начиная с директория gamedata
       2. section\_exist(<имя\_секции>) – проверка на существование секции
       3. line\_exist(<имя\_секции>,<имя строки>) – проверка на существование строки в секции
       4. line\_count(<имя\_секции>) – получение количества непустых строк в секции
       5. r\_bool(<имя\_секции>,<имя строки>) – чтение булевого типа
       6. r\_u32(<имя\_секции>,<имя строки>) – чтение u32
       7. r\_s32(<имя\_секции>,<имя строки>) – чтение s32
       8. r\_string(<имя\_секции>,<имя строки>) – чтение строки
       9. r\_float(<имя\_секции>,<имя строки>) – чтение float
       10. r\_vector(<имя\_секции>,<имя строки>) – чтение vector
       11. r\_clsid(<имя\_секции>,<имя строки>) – чтение clsid
       12. r\_token(<имя\_секции>,<имя строки>,<token\_list>) – чтение id по списку токенов
  1. Классы Lua
     1. pp\_effector

Порождён от effector, в нём переопределен метод process, в котором он изменяет параметры, служит примером того, как делать экранные эффекты.

* + 1. pp\_linear\_lerp

Порождён от effector, в нём переопределен метод process, в котором он линейно переходит от стартовых параметров да заданных.

1. Все игровые скрипты должны находится в корне директории “S:\GameData\Scripts” в файлах с расширением “.script”.
2. Есть файл настроек скриптов игры, который находится в “S:\GameData\script.ltx”. В нём есть несколько секций, каждая из которых соответствуют определённому типу игры. В каждой секции должна быть единственная строка “script, в которой через запятую должны быть указаны названия файлов, содержащих скрипты (без пути и расширения “.script”). Эти скрипты будут загружены, если будет запущен соответствующий тип игры, и будут исполняться не на клиенте, а на сервере. Секция “Common” создана для указания библиотек скриптов, которые должны быть загружены всегда, т.к. другие скрипты пользуются функциями из них. Заметьте, что эта секция должна обязательно содержать пустой элемент для загрузки скрипта, содержащегося в файле “.script”. Этот файл должен быть загружен в глобальное пространство имён (о пространствах имён читайте далее)
3. В LevelEditor в Scene\Options\Level Script можно создать секцию “level\_scripts а в ней строку “script в которой, через запятую прописать названия файлов, содержащие скрипты, которые будут загружены при загрузке этого уровня. Эти скрипты будут исполняться на клиенте, сама конфигурационная строка сохраняется в файле “level.ltx” в директории уровня.
4. Для того, чтобы скрипт был запущен, необходимо объявить в нём функцию main:

// test.script

function main()

…

end

1. Каждый скриптовый поток выполняется параллельно с другими скриптами в игре, поэтому для предотвращения объявления нескольких глобальных переменных с одинаковыми именами, каждый скрипт выполняется в своём пространстве имён (namespace), которое вычисляется исходя из названия файла, который содержит скрипт. Т.е. если скрипт называется “test.script то все его функции будут вынесены в namespace “test если же файл называется “test.test.script то namespace будет соответственно “test.test”.
2. Если вы используете в своём скрипте какие-то данные или классы из другого скрипта, вы должны указать эту зависимость, написав в основном namespace своего скрипта строку ‘module(“script\_name”)’, где script\_name - имя скрипта (без расширения)
3. Старайтесь не определять глобальные переменные вообще, только там, где это необходимо, т.к. если одна и та же функция, которая изменяет глобальные переменные, будет вызвана из разных скриптов, то получим т.н. side-affect, т.е. потенциально неуправляемое поведение скрипта.
4. Избегайте названий скриптов, которые совпадают с именами глобальных функций и классов, экспортированных в Lua, т.к. при совпадении скрипт вообще не будет загружен. Например, если попытаться запустить скрипт “sound.script в логе игры будет сообщение “! [LUA] Error : the namespace name sound is already being used by the non-table object!”
5. По консольной команде lua\_help, вы можете получить список экспортированных классов и функций.
6. Если объявить внутри скрипта функцию, то вызывать её можно как напрямую, так и указывая namespace или же используя псевдо-namespace this:

// test.script

function a()

…

end

function main()

a() --

test.a() --

this.a() --

end

1. Учтите, что скрипты могут существенно тормозить всю игру, поэтому в скриптах, работающих как отдельные скриптовые потоки, после каждого блока операций, который может забрать на себя хоть чуть-чуть процессорного времени, необходимо вызывать описанные выше модификации функции wait(…)
2. Управление объектом устроено так, что у него есть очередь заданий, функция action добавляет в очередь заданий новое. Для очистки очереди снимите управление с объекта и назначьте новое (смотрите примеры).
3. Лог Lua находится в общем логе(<user\_name>.log), каждое сообщение имеет префикс «\* [LUA] » или «! [LUA] » для сообщений об ошибках. Кроме того, есть отдельный лог LUA, который называется <user\_name>\_<lua>.log
4. Для того чтобы быстро исправить ошибку, мне необходимо стабильно повторяемый глюк на простом уровне без других скриптовых объектов, тогда локализация глюка займёт существенно меньше времени.
5. Удачи, и не подходите ко мне слишком часто, научитесь читать ошибки, понимать природу их возникновения и сразу писать безглючный код.

Дмитрий Ясенев

12.10.2003

Motivational Graphs

(Мотивационные графы)

Motivational Graphs (MG) – сравнительно новый метод принятия решений, описанный в AI Game Programming Wisdom 2, который, по мнению авторов, является чем-то средним между Decision Trees и Neural Networks.

Суть метода заключается в том, что приятие решений основывается на мотивациях NPC, в том числе и взаимопротиворечащих.

Пусть есть несколько мотиваций. У каждой мотивации могут быть подмотивации. Все мотивации составляют мотивационный граф. Этот граф является ориентированным и по своей структуре очень похож на дерево. Отличие от дерева состоит в том, что несколько узлов могут иметь одних и тех же детей, кроме того, возможно существования нескольких корней дерева. Листья нашего графа, т.е. узлы, не имеющие подмотиваций, задают цели для NPC. Т.о. задачей MG является выяснить ЧТО делать, в то время как задачей GOAP (о нём читайте далее) является выяснить КАК делать.

Итак, как же MG выбирает, какую цель должен выполнять NPC? Для этого мы проходим по всем корням графа (т.е. узлам, не имеющим входящих рёбер), давая изначально им вес 1. Все остальные узлы имеют начальный вес 0. Далее каждый корень (исходя из каких-то своих соображений) распределяет свой вес на все свои подмотивации (при этом возможна утечка “энергии”, т.е. распределённый вес может быть в сумме меньше, чем изначальный). Мы продолжаем эту процедуру рекурсивно для детей корня до тех пор, пока не доберёмся до листьев графа. Каждый раз, когда по какой-то из веток мотиваций к ним приходит вес, он суммируется. В итоге после завершения процедуры распространения весов, мы выбираем тот лист графа, который набрал максимальный вес.

Goal-Oriented Action Planning

Goal-Oriented Action Planning (GOAP) – сравнительно новый, но уже популярный метод планирования. По сути своей он похож на методы доказательства теорем, но проще, из-за специфики его применения в играх (детальное описание смотрите в AI Game Programming Wisdom 2).

Для GOAP нам необходимо задать представление мира в терминах объекта. Каждое свойство представления мира должно оцениваться evaluator-ом. Каждое действие объекта имеет список условий своего выполнения (preconditions) и эффектов, т.е. того, что мы ожидаем от действия; кроме того, каждое действие имеет свой вес. На основании этой информации, имея текущее состояние мира и целевое (т.е. состоянием мира, в котором какие-то его свойства имеют какие-то значения), можно построить последовательность действий кратчайшего веса, которая переведёт мир из текущего состояния в целевое. Действие может несколько раз встречаться в построенной последовательности. Если при заданных параметрах последовательность не может быть построена, то выполняется предыдущая последовательность действий.

Детали реализации.

В нашей реализации GOAP размер представления мира не ограничен, т.к. при построении последовательности действий оцениваются только те свойства мира, которые необходимы для нахождения оптимального решения. После построения последовательности, будет выполняться только первое действие последовательности, до тех пор, пока не изменится цель или не изменится текущее состояние мира. Если последовательность перестроилась, и её первое действие не совпадает с предыдущим, то у предыдущего вызывается метод finalize, у нового – initialize. Для выполнения действия вызывается метод execute. Если при заданных параметрах последовательность не может быть построена, то в лог пишется предупреждение об этом вместе с dump-ом текущего состояния мира (вернее, только тех свойств мира, которые были оценены при нахождении последовательности) и целевого. Действие само по себе может быть не атомарным. Таким образом, возможно построение иерархических моделей GOAP.

Дмитрий Ясенев

02.04.2004

Finite State Machines

(конечные автоматы)

Finite State Machines (FSM) или конечные автоматы – распространенная и удобная техника для программирования поведения ботов (NPC) в компьютерных играх.

В основу FSM положен принцип того, что в каждый конкретный момент времени NPC находится в некотором четко определенном состоянии. Таких состояний конечное число, и все они известны заранее. Так, например, состояниями бота могут быть: ничего не деланье, хождение по маршруту, отыгрыш звука или анимации. Особым состоянием является нахождение NPC под контролем ИИ игры.

NPC может перейти из одного состояния в другое после выполнения некоторого условия перехода. При задании условия перехода из состояния A в состояние B мы фактически определяем, при каких условиях будет осуществлен переход из одного состояния в другое. При этом переход из B в A требует определения своего собственного условия перехода. Если условия перехода, между какими-либо двумя состояниями не задано, то переход считается невозможным.

Структуру FSM можно представить в виде ориентированного графа (рис.1), вершины которого (кружочки) – состояния, а ребра (стрелочки) – условия перехода.

0

1

2

0-1

1-0

1-2

2-0

Рис.1

Пример:

Состояния:

0 – состояние солдата под контролем ИИ

1 – солдат говорит «стой стрелять буду!»

2 – солдат говорит «ушел гад»

Условия перехода:

* 1. – солдат первый раз видит актера

1. – солдат закончил говорить фразу «стой стрелять буду!» и видит актера

1-2 - солдат закончил говорить фразу «стой стрелять буду!» и потерял из виду актера

2-0 - солдат закончил говорить фразу «ушел гад»

Юрий Добронравин

01.11.2003