Логика NPC

Теория

Предполагается, что читатель этой статьи знаком с языком LUA и основами объектно-ориентированного программирования.

История

Подход к решению проблемы игрового ИИ, выбранный создателями S.T.A.L.K.E.R. (далее буду писать просто "Сталкер"), был впервые применён в 1957 году Гербертом Саймоном (Herbert Simon) и Алленом Ньюэллом (Allen Newell) в программе GPS (General Problem Solver или Универсальный Решатель Задач).

Суть этого подхода заключается в том, что сначала задача представляется в виде набора условий, набора операторов, изменяющих эти условия, и описания начального и конечного состояний. Затем осуществляется поиск последовательности операторов, переводящей начальное состояние системы в конечное.

В Сталкере подсистема поиска последовательности операторов называется «планировщик».

ИИ в Сталкере

Как видно, этот способ решения проблем не подходит для шутера, так как ситуация в игре может меняться непредсказуемым образом и построенный план решения (последовательность операторов) станет неприменимым к текущей ситуации. Поэтому планировщик запускается каждый раз при непредвиденном развитии событий и создаёт новую последовательность действий (операторов).

Условия в игре тоже вычисляются динамически. Для этого используются специальные объекты – эвалуаторы. Эвалуатор должен содержать метод evaluate(), возвращающий true, если условие выполняется и false в противном случае. Операторы также представлены как объекты. Планировщик вызывает метод initialize() при начале работы оператора, затем он периодически вызывает метод execute().

Например, можно создать эвалуатор для условия «NPC голоден», и привязать к этому условию оператор «поесть».

Планировщик будет периодически проверять это условие (вызывать метод evaluate() эвалуатора), и если оно выполняется, инициализирует и будет выполнять оператор «поесть» до тех пор, пока условие не станет ложным.

К сожалению, в большей части скриптов все возможности планировщика не используются.

Разбор настройки и работы планировщика на примере скрипта xr_kamp

Рассмотрим скрипт xr_kamp, заставляющий сталкеров сидеть у костра и рассказывать анекдоты. Настройка планировщика осуществляется в функции add_to_binder. Параметры функции: object – объект для которого настраивается планировщик (в нашем случае это сталкер), ini, scheme, section – инициализационный файл, название схемы действий, секция ини-файла (эти параметры будут

подробно разобраны в части по созданию мода), storage – таблица для хранения текущих параметров схемы действий.

Разберём, что делает эта функция.

Сначала получаем планировщик для текущего объекта (object).

```
local manager = object:motivation_action_manager()
```

Затем присваиваем идентификаторы операторов и условий элементам массива. Это сделано просто для удобства.

Идентификаторы могут иметь любое целочисленное значение, главное, чтобы они были уникальными, то есть не использовались для других операторов и условий.

```
properties["kamp_end"]=xr_evaluators_id.stohe_kamp_base+1
properties["on_position"]=xr_evaluators_id.stohe_kamp_base+2
properties["contact"]=xr_evaluators_id.stohe_meet_base+1
operators["go_position"]=xr_actions_id.stohe_kamp_base+1
operators["wait"]=xr_actions_id.stohe_kamp_base+3
```

Для каждого идентификатора условия создадим соответствующий эвалуатор и добавим его в планировщик. В данном случае это условия: «закончить ли посиделки около костра?» и «пришёл ли я на своё место у костра?»

```
manager:add_evaluator (properties["kamp_end"],
this.evaluator_kamp_end   ("kamp_end", storage, "kamp_end"))manager:add_evaluator
(properties["on_position"],
this.evaluator_on_position  ("kamp_on_position", storage, "kamp_on_position"))
```

Теперь создадим оператор «сидеть около костра, рассказывать анекдоты, жевать колбасу и т.д.». Можно было бы реализовать эти действия как набор разных операторов, выбором которых занимался бы планировщик, но автор скрипта решил сделать один сложный оператор.

```
local action = this.action_wait (object:name(),"action_kamp_wait", storage)
```

Задаем предусловия для этого оператора. Планировщик выберет этот оператор при выполнении всех условий. Всё это значит примерно следующее: я могу сидеть у костра, если:

```
action:add_precondition   (world_property(stalker_ids.property_alive, true))
```

я живой,

```
action:add_precondition   (world_property(stalker_ids.property_danger,false))
```

опасностей нет,

```
action:add_precondition   (world_property(stalker_ids.property_enemy, false))
```

врагов нет,

```
action:add_precondition   (world_property(stalker_ids.property_anomaly,false))
```

аномалий поблизости нет,

```
xr_motivator.addCommonPrecondition(action)
```

выполняются другие важные условия (игрок не собирается со мной поговорить, я не собираюсь никого бить по морде, я не ранен, я не собираюсь стрелять по вертолёту),

```
action:add_precondition   (world_property(properties["on_position"],  true))
```

я уже нахожусь около костра.

Скажем планировщику, что он должен ожидать от выполнения этого оператора. В данном случае после выполнения этого оператора условие «закончить ли посиделки около костра?» должно стать истинным. То есть если условие стало истинным, планировщик прекратит выполнение оператора.

```
action:add_effect     (world_property(properties["kamp_end"],   true))
```

Создание оператора завершено. Добавим его в планировщик.

```
manager:add_action (operators["wait"], action)
```

Эта строчка не имеет отношения к работе планировщика. Если коротко, то она позволяет объекту получать уведомления об определённых событиях (смерть NPC – вызывается метод death_callback(), попадание пули в NPC – вызывается метод hit_callback() и т.д.)

```
xr_logic.subscribe_action_for_events(object, storage, action)
```

Создаем оператор, отвечающий за доставку NPC к его месту у костра.

```
action = this.action_go_position (object:name(),"action_go_kamp", storage)
```

Добавляем предусловия, как и для предыдущего оператора.

```
action:add_precondition   (world_property(stalker_ids.property_alive, true))
action:add_precondition   (world_property(stalker_ids.property_danger,false))
action:add_precondition   (world_property(stalker_ids.property_enemy, false))
action:add_precondition   (world_property(stalker_ids.property_anomaly,false))
xr_motivator.addCommonPrecondition(action)
action:add_precondition   (world_property(properties["on_position"],  false))
```

Единственное отличие – последнее условие. Этот оператор будет выполняться только если NPC ещё не находится на своем месте у костра, то есть если функция evaluator_on_position.evaluate() возвращает false.

В результате выполнения этого действия условие «на своём ли я месте у костра?» должно стать истинным.

```
action:add_effect     (world_property(properties["on_position"],  true))
```

Создание оператора завершено. Добавляем его к планировщику.

```
manager:add_action (operators["go_position"], action)
```

Осталось ещё одна задача. Нужно запретить планировщику активировать оператор «alife», тот самый оператор, который заставляет NPC болтаться по карте, отстреливать собачек и в конце концов попадать в аномалию. Впрочем, отстрелом врагов занимается другой оператор с идентификатором stalker_ids.action_combat_planner.

Для этого мы получаем оператор «alife»

```
action = manager:action (xr_actions_id.alife)
```

И добавляем к его предусловиям следующее: условие «закончить ли посиделки у костра?» должно быть истинным.

```
action:add_precondition   (world_property(properties["kamp_end"],   true))
```

Итак, мы настроили планировщик. Посмотрим как всё это будет работать.

В некоторый момент времени гулаг, в который попал NPC, назначает ему работу: сидеть у костра. В результате условие «закончить ли посиделки у костра?» становится ложным. Планировщик видит это изменение и пытается выработать последовательность операторов, после выполнения которой, условие бы стало истинным и NPC снова бы вернулся к выполнению высокоприоритетного оператора «alife». Для выполнения этой задачи подходит оператор «посиделки у костра», но для него не выполняется условие «я на своем месте у костра». Поэтому планировщик создаёт план из двух операторов: «дойти до костра» и «посиделки у костра». Если во время выполнения одного из операторов возникнет непредвиденная ситуация (появится враг, главный герой начнёт приставать с вопросами и т.п.), то планировщик скорректирует план, добавив оператор для устранения этой непредвиденной ситуации.

Как видно система ИИ в Сталкере обладает весьма большой гибкостью, что мы и продемонстрируем при создании мода.

Модели (или схемы) поведения в Сталкере

В наборе скриптов Сталкера предусмотрена возможность объединять операторы и условия в модели поведения. Модель поведения – это набор логически связанных операторов и условий, служащих для выполнения определённой задачи. Так скрипт xr_kamp представляет собой модель поведения, состоящую из двух операторов и двух условий.

Регистрация модели поведения

Для включения новой модели поведения в набор моделей, доступных NPC, сначала необходимо её зарегистрировать. Предположим, нам нужно зарегистрировать модель поведения, описанную в скрипте actor_need_help.script. Регистрация моделей осуществляется в скрипте modules.script. Добавим туда следующие строки:

```
if actor_need_help then – в этой строке мы проверяем что наш скрипт действительно
существует
    load_scheme("actor_need_help", "actor_need_help", stype_stalker)
end
```

Первый параметр функции load_scheme задает имя файла скрипта, второй параметр – это название модели поведения, третий параметр – тип модели поведения (возможны следующие значения: stype_stalker – модель поведения NPC, stype_mobile – модель поведения монстра, stype_item – «модель поведения» физического объекта, stype_heli – модель поведения вертолёта, stype_restrictor – «модель поведения» области пространства). Скрипты для моделей поведения разных типов пишутся по-разному. Мы будем рассматривать только модели поведения NPC.

Внимание! Для успешной работы модели поведения её скрипт должен содержать функцию add_to_binder, выполняющую настройку планировщика.

Активация/деактивация модели поведения

Некоторые модели поведения применимы в любых ситуациях (например, реакция на попадание пули в NPC или реакция на появление врага). Такие модели должны активироваться/деактивироваться в функциях

enable_generic_schemes()/disable_generic_schemes() скрипта xr_logic. В случае с моделью поведения actor_need_help,

это будет выглядеть так:

1. Создаём функции set_actor_need_help и disable_scheme в нашем скрипте actor_need_help. Эти функции будут отвечать за активацию и деактивацию нашей модели поведения.

```
function set_actor_need_help(npc,ini,scheme)
  local st=xr_logic.assign_storage_and_bind(npc, ini, scheme, "actor_need_help")
  st.enabled=true
end
function disable_scheme(npc,scheme)
  local st = db.storage[npc:id()][scheme]
  if st then
    st.enabled = false
  end
end
```

2. Добавляем следующую строку в скрипт xr_logic.script после строки «if stype == modules.stype_stalker then» в функции enable_generic_schemes()

```
  actor_need_help.set_actor_need_help(npc,ini,"actor_need_help")
```

3. Добавляем следующую строку в скрипт xr_logic.script после строки «if stype == modules.stype_stalker then» в функции disable_generic_schemes()

```
  actor_need_help.disable_scheme(npc,"actor_need_help")
```

Если же модель поведения предназначена только для использования в определённых ситуациях, то достаточно выполнить шаг 1 и использовать созданные функции по мере надобности. Например, активируя эту схему через диалог с NPC (как мы и сделаем в нашем моде).

Внимание! Я максимально упростил функции активации/деактивации модели поведения. Чтобы полностью разобраться с ними, посмотрите скрипты xr_combat, xr_kamp и другие подобные.

Приоритеты моделей поведения

Некоторые модели поведения настолько важны, что должны срабатывать в любой ситуации (например, реакция на попадание пули). Для этого в скрипте xr_motivator предусмотрена функция addCommonPrecondition(action), в эту функцию можно добавить одно из условий нашей модели поведения, чтобы другие модели поведения не могли сработать при выполнении этого условия (здесь есть свои тонкости, но мы рассмотрим их позже). Предположим, что у нас есть модель поведения actor_need_help, заставляющая NPC подбежать к ГГ и вылечить его. Пусть за проверку здоровья ГГ отвечает условие с идентификатором actor_need_help.property_actor_is_wounded. Значит, если мы хотим, чтобы NPC подбегал к ГГ не обращая внимание ни на что другое, то нужно добавить следующую строчку в функцию addCommonPrecondition(action):

```
action:add_precondition(world_property(actor_need_help.property_actor_is_wounded,fals
e))
```

Эта строчка запретит выполнение всех других действий, если условие с идентификатором actor_need_help.property_actor_is_wounded станет истинным (в нашем случае это будет означать, что ГГ сильно ранен.

Конкретное значение здоровья ГГ при котором он считается сильно раненным будет определять эвалуатор этого условия).

Создаем мод

В этом разделе мы сделаем мод, позволяющий сказать дружественно настроенному NPC, чтобы он лечил главного героя во время боя.

Постановка задачи

Итак, мы хотим, чтобы дружественные NPC наконец начали приносить пользу. Для этого научим их лечить ГГ во время боя. Распишем по пунктам:

1. Нужно добавить дружественным NPC ветку в диалоге с просьбой присматривать за ГГ и лечить его, если в этом возникнет необходимость.

2. Добавить NPC модель поведения, реализующую выполнение этой просьбы.

2.1. NPC должен действовать согласно этой модели только если ГГ находится недалеко от него.

2.2. NPC не должен далеко отходить от ГГ во время боя.

2.3. Если здоровье ГГ упало ниже определённой отметки, NPC должен подойти/подбежать и вылечить ГГ.

Что потребуется для реализации

Нам придётся изменять диалоги для некоторых NPC, для этого нужно будет изменить файлы config\gameplay\character_dialogs.xml (диалоги для всех NPC), config\localization.ltx и config\system.ltx (подробнее см. статью BAC9-FLCL или Fr3nzy). Мы изменим диалоги для всех NPC, но для неподходящих NPC диалог будет отсекаться с помощью предусловия. Потребуется также добавить файлы с текстами диалогов и функции для проверки условий, используемых в диалогах.

Для включения новой модели поведения NPC нужно будет внести изменения в скрипты scripts\modules.script (регистрация моделей поведения) и scripts\xr_motivator.script (для установки высокого приоритета нашей модели). Модификации файла xr_logic.script, в котором происходит установка общих моделей поведения, не потребуется, так как мы будем активировать нашу схему поведения при выборе определённой ветки в диалоге.

Теперь решим какие условия и операторы нам понадобятся.

Условия:

1. Состояние главного героя. Если оно ниже определённого порога, то условие станет истинным. Назначим ему идентификатор property_actor_is_wounded и эвалуатор evaluator_actor_is_wounded. Далее я буду указывать идентификатор и эвалуатор в скобках через запятую.

2. Находится ли NPC достаточно близко, чтобы вылечить ГГ. (property_ready_to_heal, evaluator_ready_to_heal)

3. Есть ли у NPC аптечки. (property_has_medkit, evaluator_has_medkit)

4. Не отошёл ли NPC слишком далеко от ГГ или ГГ от NPC. (property_faraway, evaluator_faraway)

5. Находится ли ГГ достаточно близко, чтобы имело смысл помогать ему. (property_near_enough, evaluator_near_enough).

Операторы:

1. Лечить ГГ. (act_heal, action_heal)

2. Подбежать к ГГ на дистанцию, достаточную для лечения (act_run_to_actor, action_run_to_actor)

3. Крикнуть что аптечки кончились (act_no_medkit, action_no_medkit)

4. Подобраться поближе к ГГ, чтобы быть под рукой. (act_stay_close, action_stay_close).

Реализация

Я буду писать эту статью параллельно с разработкой мода, указывая все найденные ошибки. Надеюсь, это поможет другим моддерам. Чтобы отделить результаты тестирования от описания процесса разработки мода, я буду выделять свои комментарии другим шрифтом.

Диалоги

В этом моде будет всего один диалог, и довольно простой, поэтому начнём с него. Создаём файл config\gameplay\dialogs_need_help.xml. Чтобы не возиться с идентификаторами текстов попробуем обойтись без них. Начнём с простой тестовой версии:

```
<?xml version="1.0" encoding="windows-1251" standalone="yes" ?>
<game_dialogs>
    <dialog id="actor_will_need_help">
        <precondition>actor_need_help.i_am_friend</precondition>
        <phrase_list>
            <phrase id="0">
                <text>Дружище, сможешь присмотреть за мной, если стрелять начнут?
Промедольчику, там, вколоть или

перевязку сделать, если что...</text>
                <next>1</next>
            </phrase>
            <phrase id="1">
                <text>Нет проблем, конечно помогу.</text>
            </phrase>
        </phrase_list>
    </dialog>
</game_dialogs>
```

Добавляем строку с идентификатором этого диалога в config\gameplay\character_dialogs.xml:

```
<actor_dialog>actor_will_need_help</actor_dialog>
```

Дописываем имя файла диалога в config\system.ltx в секцию «dialogs». Осталось создать функцию i_am_friend. Наш скрипт с моделью поведения будет называться scripts\actor_need_help.script, заодно пропишем там и диалоговые функции.

```
function i_am_friend(actor,npc)
  return npc:relation(actor)==game_object.friend
end
```

При первом тестировании я заменил game_object.friend на game_object.neutral, чтобы не искать друзей по всей карте.

Тестируем, что у нас получилось...

Диалог работает, но вместо текста – набор значков, оказалось, я написал текст в кодировке CP866 (DOS), нужно поменять её на CP1251. Так, теперь текст в порядке.

Выбранный подход к созданию диалога оказался удачным.

Обратите внимание, вместо идентификатора текста можно вписать сам текст. Это усложнит локализацию, но уменьшит время создания диалогов.

Теперь нам понадобятся функции проверки наличия аптечек у NPC, проверки активации схемы поведения, активации/деактивации схемы поведения. Добавляем их в наш скрипт-файл.

Первый вариант скрипта я не буду приводить, чтобы не увеличивать и так большую статью. Окончательный вариант смотрите в конце статьи. Дальше по тексту идёт простейший работающий вариант.

Теперь у нас должно быть два варианта начальной фразы для активной и неактивной схемы. Придётся экспериментировать.

'Вот первый проверенный вариант:

```
<?xml version="1.0" encoding="windows-1251" standalone="yes" ?>
<game_dialogs>
    <dialog id="actor_will_need_help">
        <precondition>actor_need_help.i_am_friend</precondition>
        <phrase_list>
            <phrase id="0">
                <next>1</next>
                <next>2</next>
            </phrase>
            <phrase id="1">
                <text>Дружище, сможешь присмотреть за мной, если стрелять начнут?
Промедольчику, там, вколоть или

перевязку сделать, если что...</text>
                <precondition>actor_need_help.scheme_is_not_active</precondition>
                <next>11</next>
                <next>12</next>
            </phrase>
            <phrase id="11">
                <text>Нет проблем, конечно помогу.</text>
                <precondition>actor_need_help.npc_have_medkit</precondition>
                <action>actor_need_help.activate_scheme</action>
            </phrase>
            <phrase id="12">
```

```
                <text>Извини, друг, аптечек совсем не осталось.</text>
                <precondition>actor_need_help.npc_havent_medkit</precondition>
            </phrase>
            <phrase id="2">
                <text>Спасибо, друг, теперь я сам справлюсь.</text>
                <precondition>actor_need_help.scheme_is_active</precondition>
                <next>21</next>
            </phrase>
            <phrase id="21">
                <text>Да не за что, ты мне помог, я - тебе.</text>
                <action>actor_need_help.deactivate_scheme</action>
            </phrase>
        </phrase_list>
    </dialog>
</game_dialogs>
```

При первом запуске игра вылетела без сообщений об ошибках. Я внимательно просмотрел все файлы и оказалось, что в скриптах вместо комментария ‘--‘ (два минуса) я поставил просто минус (рекомендую пользоваться компилятором с [www.lua.org www.lua.org] для проверки корректности скриптов). После исправления ошибки игра запустилась, но диалог так и не появился. Небольшая дискуссия на форуме (спасибо Z.E.N. и Arhet) показала, что придётся сделать два диалога.

Кроме того, при тестах первых вариантов, выяснилось, что всегда выбирается вариант диалога с просьбой о помощи. То есть схема поведения не активируется.

Как оказалось, в качестве параметров в функции передаются не объекты ГГ и NPC, а объекты говорящего в данный момент персонажа и его собеседника. То есть, если фраза принадлежит NPC, то первый параметр будет объектом для NPC, а не для ГГ. Поэтому я изменил названия параметров и переписал функции.

Итак, все проблемы решены. Файл диалога принял следующий вид (config\gameplay\dialogs_need_help.xml):

```
<?xml version="1.0" encoding="windows-1251" standalone="yes" ?>
<game_dialogs>
    <dialog id="actor_will_need_help">
        <precondition>actor_need_help.i_am_friend</precondition>
        <precondition>actor_need_help.scheme_is_not_active</precondition>
        <phrase_list>
            <phrase id="0">
                <text>Дружище, сможешь присмотреть за мной, если стрелять начнут?
Промедольчику, там, вколоть или

перевязку сделать, если что...</text>
                <next>11</next>
                <next>12</next>
            </phrase>
            <phrase id="11">
                <precondition>actor_need_help.npc_have_medkit</precondition>
                <text>Нет проблем, конечно помогу.</text>
                <action>actor_need_help.activate_scheme</action>
            </phrase>
            <phrase id="12">
                <precondition>actor_need_help.npc_havent_medkit</precondition>
                <text>Извини, друг, аптечек совсем не осталось.</text>
            </phrase>
        </phrase_list>
    </dialog>
    <dialog id="actor_will_not_need_help">
        <precondition>actor_need_help.i_am_friend</precondition>
        <precondition>actor_need_help.scheme_is_active</precondition>
        <phrase_list>
```

```
            <phrase id="0">
                <text>Спасибо, друг, теперь я сам справлюсь.</text>
                <next>21</next>
            </phrase>
            <phrase id="21">
                <text>Да не за что, ты мне помог, я - тебе.</text>
                <action>actor_need_help.deactivate_scheme</action>
            </phrase>
        </phrase_list>
    </dialog>
</game_dialogs>
```

Функции, поддерживающие работу диалога, теперь выглядят так (файл scripts\actor_need_help.script):

```
function i_am_friend(talker,target)
  return target:relation(talker)==game_object.friend
end

-- За основу этой функции взята функция dialogs.actor_have_medkit
function npc_have_medkit(talker, target)
  return talker:object("medkit") ~= nil or
       talker:object("medkit_army") ~= nil or
       talker:object("medkit_scientic") ~= nil
end
function npc_havent_medkit(talker, target)
  return not npc_have_medkit(talker,target)
end

-- Так как модель поведения еще не написана, вставим заглушки
local scheme_status={}
function scheme_is_active(talker,target)
  return scheme_status[target:id()]==true -- сравниваем с true, чтобы функция не
возвращала nil
end
function scheme_is_not_active(talker,target)
  return not scheme_is_active(talker,target)
end
function activate_scheme(talker,target)
  scheme_status[talker:id()]=true
end
function deactivate_scheme(talker,target)
  scheme_status[talker:id()]=nil -- присваиваем nil, чтобы освободить память, занятую
этим элементом массива
end
```

И в файл config\gameplay\character_dialogs.xml добавлены строки:

```
<actor_dialog>actor_will_need_help</actor_dialog>
<actor_dialog>actor_will_not_need_help</actor_dialog>
```

В результате получился работающий диалог, но NPC выглядит просто как ходячая аптечка. В окончательном варианте, я добавил некоторые «человеческие» реакции. Да и сама модель поведения пока отсутствует - вместо неё стоят заглушки. Исправим это.

Модель поведения

Начнём создание модели поведения с разработки эвалуаторов. Эвалуатор должен представлять собой объект класса унаследованного от класса property_evaluator.

Возьмём для начала эвалуатор evaluator_faraway определяющий, что NPC находится слишком далеко от ГГ. Этот эвалуатор требуется для того, чтобы NPC не отходил слишком далеко от ГГ и мог в случае надобности быстро подбежать к нему и оказать помощь.

Объявляем класс эвалуатора:

```
class "evaluator_faraway" (property_evaluator)
```

Определяем функцию инициализации (в LUA это аналог конструктора объекта)

```
function evaluator_faraway:__init(name, storage) super (nil, name)
  self.st = storage
end
```

Ключевое слово «super» служит для вызова конструктора базового класса. Член «st» будет хранить ссылку на таблицу состояния нашей модели поведения.

Теперь нужно определить функцию evaluate(), ради которой и создавался эвалуатор. По-видимому всё просто, нужно проверить расстояние от NPC до ГГ и вернуть true, если это расстояние больше определённого значения. Но давайте подумаем. Когда эвалуатор возвратит true, заработает оператор, заставляющий NPC подойти поближе к ГГ, то есть расстояние моментально уменьшится и эвалуатор начнёт возвращать false, что приведёт к переходу NPC под управление игрового ИИ. ИИ может опять решить удалиться от ГГ, что приведёт к повторному срабатыванию эвалуатора. В результате возникнет замкнутый цикл, и NPC будет крутиться на одном месте (на самом деле этот цикл рано или поздно разорвётся из-за изменения игровой ситуации, но лучше вообще избежать его).

Можно использовать разные пути для решения этой проблемы. Попробуем сделать так: будем использовать два расстояния, эвалуатор сработает при достижении первого и будет оставаться активным, пока расстояние не станет меньше второго.

```
local min_faraway_dist=10
local max_faraway_dist=20

function evaluator_faraway:evaluate()
  local actor=db.actor
  if not actor then
  -- ГГ ещё не заспаунился
    return false
  end
  local dist=actor:position():distance_to(self.object:position())
  if dist>max_faraway_dist then
    self.st.faraway=true
  elseif dist<min_faraway_dist then
    self.st.faraway=false
  end
  return self.st.faraway==true
end
```

Эвалуатор готов, но нужно как-то его протестировать. Поэтому давайте создадим минимальную модель поведения из одного условия и одного оператора. Нам нужен оператор, перемещающий NPC поближе к ГГ. Объявляем класс action_stay_close, унаследованный от action_base, и определяем его конструктор.

```
class "action_stay_close" (action_base)

function action_stay_close:__init(name, storage) super (nil, name)
  self.st=storage
end
```

Оператор должен содержать функции initialize(), execute() и, возможно, finalize().

Функция initialize() вызывается при начала работы оператора, то есть в момент, когда планировщик ставит этот оператор в первую позицию плана.

```
function action_stay_close:initialize()
  local npc=self.object
  -- Не знаю зачем эти две функции, но они используются во всех операторах
  npc:set_desired_position()
  npc:set_desired_direction()
  -- Сбрасываем текущие анимации
  npc:clear_animations()
  -- Задаём параметры движения
  npc:set_detail_path_type(move.line)
  npc:set_body_state(move.standing)
  npc:set_movement_type(move.run)
  npc:set_path_type(game_object.level_path)
  -- Эксперименты показали, что эта функция устанавливает скорость движения
(anim.danger -
минимальная скорость, anim.free - нормальная, anim.panic - максимальная)
  npc:set_mental_state(anim.panic)
  -- Повышаем зоркость NPC
  npc:set_sight(look.danger, nil, 0)
  -- Освободим сталкера от всех идиотских ограничений
  npc:remove_all_restrictions()
-- Зададим смещение точки назначения, чтобы помощники не сбивались в кучу
  self.offset=vector():set(math.random()*6-3,0,math.random()*6-3)
  self.offset:normalize()
end
```

Функция execute() периодически вызывается во время выполнения оператора. Частота вызовов, по-видимому, зависит от расстояния от NPC до ГГ.

```
function action_stay_close:execute()
  local npc=self.object
  local actor=db.actor
  if not actor then
  -- Хм, что-то не так. Может быть ГГ перешёл на другой уровень? Запрещаем схему
поведения
    self.st.enabled=false
  end
  -- Получаем ближайшую доступную точку в 5 метрах от ГГ
  -- Сначала, я попробавал использовать функцию npc:vertex_in_direction, но она не
работает
  local vertex_id=level.vertex_in_direction(actor:level_vertex_id(),self.offset,5)
  local act_v_id=actor:level_vertex_id()
  -- Отправляем нашего NPC в найденную точку
  local acc_id=utils.send_to_nearest_accessible_vertex( npc, vertex_id )
  if self.st.dist and self.st.dist>max_faraway_dist then
-- если NPC находится слишком далеко от ГГ пусть пробежиться побыстрее
    npc:set_mental_state(anim.panic)
  else
    npc:set_mental_state(anim.free)
  end
end
```

Функция настройки планировщика.

```
function add_to_binder(object, char_ini, scheme, section, st)
  local manager = object:motivation_action_manager()
  local property_wounded = xr_evaluators_id.sidor_wounded_base
```

```
  -- Удаляем эвалуатор, так как в xr_motivator мы установили его в
property_evaluator_const
  manager:remove_evaluator(property_faraway)
  -- и заменяем его нашим
  manager:add_evaluator(property_faraway, evaluator_faraway("evaluator_faraway",st))
    -- Создаём оператор
  local action = action_stay_close("action_stay_close",st)
    -- и настраиваем предусловия. 1. Сталкер жив
  action:add_precondition(world_property(stalker_ids.property_alive, true))
  -- 2. Сталкер не ранен
  action:add_precondition(world_property(property_wounded, false))
  -- Я использую свой мод для обхода аномалий, иначе от помощников мало толку.
  if anomaly_evader then
  -- 3. Рядом нет аномалий
    action:add_precondition (world_property(1099,false))
  end
 -- 4. Сталкер слишком далеко от ГГ
   action:add_precondition(world_property(property_faraway, true))
  action:add_effect (world_property(property_faraway, false))
  -- Добавляем оператор в планировшик
  manager:add_action (act_stay_close, action)
  -- Теперь подкорректируем стандартные операторы, чтобы помощник не отвлекался на
всякую ерунду.
  action=manager:action(stalker_ids.action_alife_planner)
  action:add_precondition(world_property(property_faraway, false))
  action=manager:action(stalker_ids.action_combat_planner)
  action:add_precondition(world_property(property_faraway, false))
  action=manager:action(stalker_ids.action_danger_planner)
  action:add_precondition(world_property(property_faraway, false))
end
```

Добавим функции активации/деактивации схемы поведения.

```
function set_help(npc, ini)
  local st = xr_logic.assign_storage_and_bind(npc, ini, "actor_need_help")
  st.enabled=true
end
function disable_scheme(npc, scheme)
  local st = db.storage[npc:id()][scheme]
  if st then
    st.enabled = false
  end
end
```

Изменим диалоговые функции-заглушки.

```
function activate_scheme(talker,target)
  set_help(talker,talker:spawn_ini())
  scheme_status[talker:id()]=true
end
function deactivate_scheme(talker,target)
  disable_scheme(talker,"actor_need_help")
  scheme_status[talker:id()]=nil
end
```

Добавим в функцию xr_motivator.addCommonPrecondition() следующие строки, чтобы заблокировать стандартные схемы поведения.

```
  if actor_need_help then
    action:add_precondition (world_property(actor_need_help.property_faraway,false))
  end
```

Если попробовать запустить мод сейчас, то игра просто вылетит. Причина в том, что мы добавили предусловие для стандартных схем поведения, но не добавили эвалуатор этого условия. Поэтому добавляем в функцию xr_motivator.net_spawn() следующие строки:

```
    local manager = self.object:motivation_action_manager()
    if actor_need_help then
      manager:add_evaluator(actor_need_help.property_faraway,
property_evaluator_const(false))
    end
```

Для того чтобы снизить нагрузку на процессор, используем property_evaluator_const, который всегда возвращает одно и тоже значение. В результате тестирования выяснилось, что не все NPC подчиняются нашей схеме поведения. Причины этого пока не ясны, требуется дополнительное тестирование и очень желательна помощь разработчиков (хотя бы для того чтобы узнать как выяснить какой оператор действует в данный момент на NPC).

Файлы мода

Мод можно скачать здесь.

Ссылка обновлена. Выложена новая версия. Причиной "зависаний" NPC был конфликт со state_mgr.script. Код мода, приведённый в статье, не устанавливал состояние NPC с помощью state_mgr.set_state(), в результате state_mgr пытался вернуть состояние NPC к начальному (до срабатывания нашей модели поведения), что и приводило к "зависаниям".

Более подробное описание мода смотри здесь.

Спасибо за внимание, **Red75**.