Управление персонажем

Материал из xrWiki

Движок Сталкера реализует две концепции управления объектами: FSM (Finite State Machines) и GOAP (Goal-Oriented Action Planning). Начнем с FSM.

О том, что такое FSM, читаем у Ясенева:

> *"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."*

Для реализации управления объектами с помощью концепции FSM в классе game_object есть ряд методов:

```
    function see(const game_object*);
    function command(const entity_action*, boolean);
    function action_by_index(number);
    function action_count() const;
    function action() const;
    function get_script() const;
    function get_script_name() const;
    function reset_action_queue();
    function script(boolean, string);
    function can_script_capture() const;
```

Они подробно будут разобраны далее. Методы эти можно вызывать для объектов, наследующих от класса CScriptEntity, или, проще говоря - для:

- торговцев,
- сталкеров,
- монстров,
- машин,
- прожекторов.

Не всякий объект, даже из перечисленных групп, подходит для управления. Это вызвано тем, что неписями постоянно кто-то рулит - движок, смарт-террейны, игровая логика. Однако, выход есть. В классе game_object существует метод script():

```
    void script(bool f, string g);
```

Этот метод позволяет взять (или отпустить) объект под скриптовый контроль принудительно. Аргумента тут два: первый - булевое значение, определяющее, взять или отпустить объект из-

под контроля. Второй - имя управляющего скрипта. Может быть абсолютно любым, это сделано, видимо, для отладки, чтобы можно было видеть, в каком состоянии находится непись.

Также существуют вспомогательные методы для определения объекта под скриптом:

```
    bool get_script() const;
        //возвращает true, если объект контролируется скриптом.
    string get_script_name() const;
        //возвращает имя скрипта, контролирующего объект. Если нет - возвращает "".
    bool can_script_capture() const;
        //возвращает true, если объект можно взять под скриптовый контроль.
```

Итак, объект взят под контроль. В таком состоянии объект становится слеп и глух ко всем раздражителям игрового мира (не реагирует даже на стрельбу). Чтобы прояснить, как им управлять, следует сказать пару слов о специфике скриптового управления.

У каждого объекта, который можно взять под скрипт, существует очередь СОСТОЯНИЙ, через которые он проходит. У объектов не под скриптом очередь пуста. Для управления очередью доступны следующие методы:

```
    int action_count() const;
        //возвращает размер очереди состояний объекта. Для объектов без очереди возвращает 0.
    void reset_action_queue();
        //завершает текущее состояние и очищает очередь
    entity_action* action_by_index(int action_index);
        //возвращает объект состояния по индексу, передаваемому аргументом action_index. Если не найден - 
вернет nil
    entity_action* action() const;
        //возвращает объект текущего состояния
    void command(const entity_action* tpEntityAction, bool bHighPriority);
        //добавляет в очередь состояний объект состояния tpEntityAction. Аргумент bHighPriority 
определяет приоритет добавляемого состояния. Если true, текущее принудительно завершается.
```

В целом, можно свободно управлять очередью состояний. Дело за малым - разобраться, как эти состояния делать. Каждое состояние - объект класса entity_action (CScriptEntityAction в движке). Вот его псевдоописание:

```
class entity_action {

//конструкторы
    entity_action* entity_action();
    entity_action* entity_action(const entity_action*);

//функции установки конкретного действия.
    void set_action(act&);
    void set_action(anim&);
    void set_action(cond&);
    void set_action(look&);
    void set_action(move&);
    void set_action(object&);
    void set_action(particle&);
    void set_action(sound&);

//общие функции
    bool completed();
    bool all();
        //проверяет, есть ли хоть одно незавершенное действие и выдает false, если есть. Зачем две 
одинаковых функции - неясно.
    bool time();
        //проверяет, закончилось ли время жизни действия
```

```
//функции проверки окончания экшена (флаг m_bCompleted)
    bool anim() const;
    bool look() const;
    bool move() const;
    bool object() const;
    bool particle() const;
    bool sound() const;
};
```

Создание состояния можно описать так:
1. Создание объекта состояния методом entity_action()
2. Заполнение состояния действиями с помощью методов set_action(). Вот список доступных действий:

```
    -act
        //глобальное поведение монстров
    -anim
        //проигрывание анимации монстрами/сталкерами
    -look
        //задание направления взгляда монстра/сталкера
    -move
        //движение объекта
    -object
        //взятие в руки предмета
    -particle
        //проигрывание партикла на монстре/сталкере
    -sound
        //проигрывание звука от монстра/сталкера
    -cond
        //не действие, по сути, а объект, управляющий переходом в следующее состояние. С помощью задания cond можно управлять длительностью состояний.
```

Перед этим надо создать объект соответствующего ДЕЙСТВИЯ (как - смотрите ниже). В одном состоянии может быть несколько РАЗНЫХ действий.

Далее состояние можно помещать в очередь состояний объекта. Следует заметить, что удалить действие из состояния нельзя, так что создавайте состояния с умом. Впрочем, всегда можно создать очередь из пустых состояний, а потом заполнять их как угодно.

Описания классов действий:

```
class act {                 //глобальное поведение монстров
//MonsterSpace::EScriptMonsterGlobalAction
    const rest = 0;         //отдыхать. Место отдыха монстр выбирает сам. У меня ложился под дерево.
    const eat = 1;          //кушать труп.
    const attack = 2;       //атаковать что-нибудь
    const panic = 3;        //убегать в панике

//конструкторы:
    void act();
        //кто его знает, зачем такой конструктор - ни одно свойство у класса выставить нельзя.
    void act(enum MonsterSpace::EScriptMonsterGlobalAction);
        //создает экшен с установленным типом поведения. Этот конструктор можно вызвать только с типом rest. attack и eat просто не будут работать, вместо panic монстр уходить спокойно погулять.
    void act(enum MonsterSpace::EScriptMonsterGlobalAction, game_object*);
        //Конструктор для eat, attack, panic. Вторым аргументом задается объект, по отношению к которому устанавливается поведение. Для eat - это объект трупа, для attack - кого атакуем, для panic - от кого бежим.

};
```

```
class anim {                                            //анимация сталкеров и монстров.
//MonsterSpace::EMentalState
    const danger = 0;                                   //состояние опасности
    const free = 1;                                     //обычное состояние
    const panic = 2;                                    //состояние паники

//MonsterSpace::EScriptMonsterAnimAction
    const stand_idle = 0;                       //спокойно стоять
    const sit_idle = 1;                                 //спокойно сидеть
    const lie_idle = 2;                                 //спокойно лежать
    const eat = 3;                                      //есть
    const sleep = 4;                                    //спать
    const rest = 5;                                     //отдыхать
    const attack = 6;                                   //атаковать
    const look_around = 7;                      //вертеть башкой
    const turn = 8;                                     //повернуться

//конструкторы
    void anim ();
        //создает пустой объект экешна
    void anim (string caAnimationToPlay);
    void anim (string caAnimationToPlay, bool use_single_hand);
        //создает экешен с установленной анимацией
//конструкторы для сталкеров
    void anim (enum MonsterSpace::EMentalState);
        //создает пустой объект экешна с установленным состоянием
        //!ПРОВЕРИТЬ! судя по всему, сам по себе не работает, надо использовать в паре с move
//конструктор для монстров
    void anim (enum MonsterSpace::EScriptMonsterAnimAction, int index);
        //создает пустой объект экешна с установленным состоянием и индексом анимации (?). Задавал 0, все
работало

//методы
    bool completed();
        //закончен ли экшн
    void type(enum MonsterSpace::EMentalState state);
        //установить состояние
    void anim(string caAnimationToPlay);
        //установить анимацию
};

class cond {                                            //установка времени действия экшена
//CScriptActionCondition::EActionFlags
    const move_end = 1;
    const look_end = 2;
    const anim_end = 4;
    const sound_end = 8;
    const object_end = 32;
    const time_end = 64;
    const act_end = 128;

//такое чувство, что не работает ничего, кроме time_end.

//конструкторы.
    void cond ();
        //создает пустой объект. Непонятно, зачем.
    void cond (int dwFlags);
        // создает объект, устанавливая флаги экшенов.
    void cond (int dwFlags, double dTime);
        // создает объект, устанавливая флаги экшенов и время жизни экшена.

};

class look {                                            //установка взгляда сталкеров
//SightManager::ESightType
    const cur_dir = 0;                                  //смотреть по текущему направлению
    const path_dir = 1;                                 //смотреть по пути
    const direction = 2;                        //смотреть в каком-то направлении
    const point = 3;                                    //смотреть в точку только головой
//      eSightTypeObject  = 4
```

```
    const danger = 5;                                   //смотреть по сторонам куда хочу, даже если придётся
менять направление тела
    const search = 6;                                   //смотреть по сторонам так, чтобы не менять
направление тела, которое ориентировано по пути
//      eSightTypeLookOver  = 7
//      eSightTypeCoverLookOver  = 8
//      eSightTypeFireObject  = 9
    const fire_point = 10;                  //смотреть в точку головой и автоматом

    void look ();
        //создает пустой завешенный (!) экшн
//для cur_dir и path_dir.
    void look (enum SightManager::ESightType tWatchType);
//для остальных типов
    void look (enum SightManager::ESightType tWatchType, vector& direction);
        //tWatchType - тип взгляда, direction - точка, куда смотрим.
    void look (enum SightManager::ESightType tWatchType, game_object* tpObjectToWatch);
    void look (enum SightManager::ESightType tWatchType, game_object* tpObjectToWatch, string
bone_to_watch);
        //tWatchType - тип взгляда, tpObjectToWatch - объект, на который смотрим, bone_to_watch - кость,
на которую смотрим
//по идее, тут можно задать скорость поворота, но у меня оба конструктора не работают.
    void look (const vector& target, float vel1, float vel2);
    void look (game_object* tpObjectToWatch, float vel1, float vel2);

    bool completed();
        //закончен ли экшн
    void type(enum SightManager::ESightType);
        //установить отип взгляда.
    void object(game_object* tpObjectToWatch);
        //установить объект, на который смотрим.
    void bone(string);
        //установить кость, на которую смотрим.
    void direct(const vector&);
        //установить направление взгляда.
};

class move {                                            //движение объектов. Нельзя применять для
прожекторов.
//MonsterSpace::EBodyState
    const crouch = 0;
    const standing = 1;
//MonsterSpace::EMovementType
    const walk = 0;
    const run = 1;
    const stand = 2;
//DetailPathManager::EDetailPathType
    const curve = 0;
    const dodge = 1;
    const criteria = 2;
    const line = 0;
    const curve_criteria = 2;
//CScriptMovementAction::EInputKeys
    const none = 1;
    const fwd = 2;
    const back = 4;
    const left = 8;
    const right = 16;
    const up = 32;                  //ShiftUp
    const down = 64;                //ShiftDown
    const handbrake = 128;
    const on = 256;                 //EngineOn
    const off = 512;                //EngineOff
//MonsterSpace::EScriptMonsterMoveAction
    const walk_fwd = 0;
    const walk_bkwd = 1;                                //никто из монстров не умеет бегать назад. Зачем это?
    const run_fwd = 2;
    const drag = 3;
    const jump = 4;
    const steal = 5;
```

```
//MonsterSpace::EScriptMonsterSpeedParam
    const default = 0;
    const force = 1;
//      eSP_None = -1

//конструктор объекта с дефолтными параметрами
    void move ();
//конструкторы для машин
    void move (enum CScriptMovementAction::EInputKeys tInputKeys);
    void move (enum CScriptMovementAction::EInputKeys tInputKeys, float fSpeed);
        //с InputKeys все понятно, fSpeed - ограничение скорости. По умолчанию 0, то есть, нет
ограничения
//конструкторы для сталкерья
    void move (enum MonsterSpace::EBodyState tBodyState, enum MonsterSpace::EMovementType tMovementType,
enum DetailPathManager::EDetailPathType tPathType, game_object* tpObjectToGo);
    void move (enum MonsterSpace::EBodyState tBodyState, enum MonsterSpace::EMovementType tMovementType,
enum DetailPathManager::EDetailPathType tPathType, game_object* tpObjectToGo, float fSpeed);
        //tBodyState - в каком положении идти (сидя, стоя), tMovementType - как идти (бежать, стоять,
идти), tPathType - тип пути (прямая, кривая, дуга), tpObjectToGo - за кем идти, fSpeed - с какой
скоростью идти.
    void move (enum MonsterSpace::EBodyState tBodyState, enum MonsterSpace::EMovementType tMovementType,
enum DetailPathManager::EDetailPathType tPathType, const patrol& tPatrolPathParams);
    void move (enum MonsterSpace::EBodyState tBodyState, enum MonsterSpace::EMovementType tMovementType,
enum DetailPathManager::EDetailPathType tPathType, const patrol& tPatrolPathParams, float fSpeed);
        //см выше, только вместо объекта - путь, по которому идем.
    void move (enum MonsterSpace::EBodyState tBodyState, enum MonsterSpace::EMovementType tMovementType,
enum DetailPathManager::EDetailPathType tPathType, const vector& position);
    void move (enum MonsterSpace::EBodyState tBodyState, enum MonsterSpace::EMovementType tMovementType,
enum DetailPathManager::EDetailPathType tPathType, const vector& position, float fSpeed);
        //см выше, вместо объекта - координаты, куда идем.
//конструкторы для монстров.
    void move (enum MonsterSpace::EScriptMonsterMoveAction tAct, patrol& tPatrolPathParams);
    void move (enum MonsterSpace::EScriptMonsterMoveAction tAct, patrol& tPatrolPathParams, float
dist_to_end);
    void move (enum MonsterSpace::EScriptMonsterMoveAction tAct, patrol& tPatrolPathParams, float
dist_to_end, enum MonsterSpace::EScriptMonsterSpeedParam speed_param);
    void move (enum MonsterSpace::EScriptMonsterMoveAction tAct, vector& tPosition);
    void move (enum MonsterSpace::EScriptMonsterMoveAction tAct, vector& tPosition, float fSpeed);
    void move (enum MonsterSpace::EScriptMonsterMoveAction tAct, vector& tPosition, float fSpeed, enum
MonsterSpace::EScriptMonsterSpeedParam);
    void move (enum MonsterSpace::EScriptMonsterMoveAction tAct, int node_id, vector& tPosition);
    void move (enum MonsterSpace::EScriptMonsterMoveAction tAct, int node_id, vector& tPosition, float
dist_to_end);
    void move (enum MonsterSpace::EScriptMonsterMoveAction tAct, game_object* tpObjectToGo);
    void move (enum MonsterSpace::EScriptMonsterMoveAction tAct, game_object* tpObjectToGo, float
dist_to_end);
    void move (enum MonsterSpace::EScriptMonsterMoveAction tAct, game_object* tpObjectToGo, float
dist_to_end, enum MonsterSpace::EScriptMonsterSpeedParam speed_param);

    void move (const vector& tPosition, float dist_to_end);

    bool completed();
        //закончен ли экшн

//методы установки параметров объекта
    void body(enum MonsterSpace::EBodyState tBodyState);
        //состояние тела
    void move(enum MonsterSpace::EMovementType tMovementType);
        //тип движения
    void path(enum DetailPathManager::EDetailPathType tPathType);
        //тип пути
    void object(game_object* tpObjectToGo);
        //к кому идти
    void patrol(const CPatrolPath *path, shared_str path_name);
        //путь для движения
    void position(const vector& tPosition);
        //координаты назначения
    void input(enum CScriptMovementAction::EInputKeys tInputKeys);
        //клавиши управления
};
```

```
class object {                                          //манипуляции сталкеров с объектом (оружие,
рации и типа того). Также работает для машин (открыть/закрыть дверь).
//MonsterSpace::EObjectAction
    const switch1 = 0;
    const switch2 = 1;
    const reload = 2;
    const reload1 = 2;
    const reload2 = 3;
    const aim1 = 4;
    const aim2 = 5;
    const fire1 = 6;
    const fire2 = 7;
    const idle = 8;
    const strap = 9;
    const drop = 10;
//      eObjectActionAimReady1  = 11
//      eObjectActionAimReady2  = 12
//      eObjectActionAimForceFull1  = 13
//      eObjectActionAimForceFull2  = 14
    const activate = 15;
    const deactivate = 16;
    const use = 17;
    const turn_on = 18;
    const turn_off = 19;
    const show = 20;
    const hide = 21;
    const take = 22;
//      eObjectActionMisfire1  = 23
//      eObjectActionEmpty1  = 24
//      eObjectActionNoItems  = 65535
    const dummy = -1;

//конструкторы
    void object ();
//конструкторы для сталкеров.
    void object (game_object* tpLuaGameObject, enum MonsterSpace::EObjectAction tObjectActionType);
    void object (game_object* tpLuaGameObject, enum MonsterSpace::EObjectAction tObjectActionType, int
dwQueueSize);
        //tpLuaGameObject – объект, с которым нужно что-то делать, tObjectActionType - что делать,
dwQueueSize – размер очереди при стрельбе
//конструктор для машин
    void object (string caBoneName, enum MonsterSpace::EObjectAction tObjectActionType);
        //caBoneName - имя кости (например, для того, чтобы открыть/закрыть дверь в машине
(activate\deactivate)). Но у меня открыть/закрыть двери не вышло.
//это не работает
    void object (enum MonsterSpace::EObjectAction tObjectActionType);

    bool completed();
        //закончен ли экшн
    void object(string caBoneName);
        //установить кость, к которой присоединяем объект
    void object(game_object* tpLuaGameObject);
        //установить объект, который присоединяем
    void action(enum MonsterSpace::EObjectAction tObjectActionType);
        //установить тип экшена
};

class particle {                                    //отыгрывание на объекте партиклов

//конструкторы
    particle ();
    particle (string caPartcileToRun, string caBoneName);
    particle (string caPartcileToRun, string caBoneName, const particle_params& tParticleParams);
    particle (string caPartcileToRun, string caBoneName, const particle_params& tParticleParams, bool
bAutoRemove);
        //caPartcileToRun – имя системы частиц, caBoneName – имя кости, tParticleParams – экземпляр
класса particle_param, bAutoRemove – зацикленный партикл. Первые два параметра обязательны, остальные по
умолчанию равны нулю
    particle (string caPartcileToRun, const particle_params& tParticleParams);
    particle (string caPartcileToRun, const particle_params& tParticleParams, bool bAutoRemove);
```

```
        //см. выше, точка проигрывания партикла берется из tParticleParams.

    bool completed();
        //закончен ли экшн
    void set_velocity(const vector& vel);
        //установить скорость движения партикла
    void set_position(const vector& pos);
        //установить координаты партикла
    void set_bone(string caBoneName);
        //установить кость, от которой играется партикл
    void set_angles(const vector& angle);
        //установить углы (?) партикла
    void set_particle(string caParticleToRun, bool bAutoRemove);
        //установить партикл
};

class particle_params {                          //параметры отыгрывания партиклов
    particle_params ();
    particle_params (const vector& tPositionOffset);
    particle_params (const vector& tPositionOffset, const vector& tAnglesOffset);
    particle_params (const vector& tPositionOffset, const vector& tAnglesOffset, const vector&
tVelocity);
};

class patrol {                                          //параметры движения сталкеров и монстров по
путям. Используется в классе move.
//PatrolPathManager::EPatrolStartType
    const start = 0;
//      ePatrolStartTypeLast  = 1
    const nearest = 2;
    const custom = 3;               //ePatrolStartTypePoint
    const next = 4;
//PatrolPathManager::EPatrolRouteType
    const stop = 0;
    const continue = 1;
//common
    const dummy = -1;

    patrol (string caPatrolPathToGo);
    patrol (string caPatrolPathToGo, enum PatrolPathManager::EPatrolStartType tPatrolPathStart);
    patrol (string caPatrolPathToGo, enum PatrolPathManager::EPatrolStartType tPatrolPathStart, enum
PatrolPathManager::EPatrolRouteType tPatrolPathStop);
    patrol (string caPatrolPathToGo, enum PatrolPathManager::EPatrolStartType tPatrolPathStart, enum
PatrolPathManager::EPatrolRouteType tPatrolPathStop, bool bRandom);
    patrol (string caPatrolPathToGo, enum PatrolPathManager::EPatrolStartType tPatrolPathStart, enum
PatrolPathManager::EPatrolRouteType tPatrolPathStop, bool bRandom, int index);
    //аргументы конструкторов: <имя пути>, <тип старта пути>, <тип финиша пути>, <если путь имеет
ветвления, то выбирать ли их случайно>. Нетрудно видеть, что первый аргумент конструктора (имя пути) -
обязательный, остальные по умолчанию таковы: тип старта = patrol.nearest, тип финиша = patrol.continue,
случайность = true.

    int level_vertex_id(int point_id) const;
        //возвращает число заданной точки пути. Аргумент - номер точки, начиная с 0 (pN:level_vertex_id =
... в алл.спавне).
    vector point(int point_id);
        //возвращает координаты заданной точки пути. Аргумент - номер точки, начиная с 0 (pN:position =
... в алл.спавне).
    bool flag(int point_id, int flag) const;
        //возвращает true, если значение флага в точке number1 равно number2, иначе false. Агрументы:
первый - номер точки, второй - число (в оригинальных скриптах почему-то только от 1 до 32)
    int game_vertex_id(int point_id) const;
        //возвращает гейм вертекс заданной точки пути. Аргумент - номер точки, начиная с 0
(pN:game_vertex_id = ... в алл.спавне).
    flags32 flags(int point_id) const;
        //возвращает флаг данной точки пути (pN:flags = ... в алл.спавне). Аргумент - номер точки,
начиная с 0. Значение флага можно узнать методом get() класса flags32
    string name(int point_id) const;
        //возвращает название заданной точки пути. Аргумент - номер точки, начиная с 0 (pN:name = ... в
алл.спавне).
    int index(string name) const;
```

```
        //возвращает id точки пути с заданным именем.
    bool terminal(int point_id) const;
        // возвращает true, если из данной точки нет переходов на другие точки и false, если есть (есть
ли pN:links = ... в алл.спавне). Аргумент - номер точки, начиная с 0.
    int count() const;
        //возвращает количество точек в пути
    int get_nearest(const vector& pos) const;
        //ищет ближайшую к заданным вектором координатам точку пути
};

class sound {                                           //отыгрывание объектом звуков
//MonsterSound::EType
//     eMonsterSoundBase  = 0
    const idle = 1;
    const eat = 2;
    const attack = 3;                       //eMonsterSoundAggressive
    const attack_hit = 4;
    const take_damage = 5;
//     eMonsterSoundStrike  = 6
    const die = 7;
//     eMonsterSoundDieInAnomaly  = 8
    const threaten = 9;
    const steal = 10;
    const panic = 11;
//     eMonsterSoundIdleDistant  = 12
//     eMonsterSoundScript  = 128
//     eMonsterSoundCustom  = 16384
//     eMonsterSoundDummy  = -1

        //видимо, не экспортированы, но есть
    void sound (string caSoundToPlay, const vector& tPosition, const vector& tAngleOffset, bool bLooped,
enum MonsterSound::EType sound_type);
    void sound (string caSoundToPlay, string caBoneName, const vector& tPositionOffset, const vector&
tAngleOffset, bool bLooped, enum MonsterSound::EType sound_type);
    void sound (sound_object& sound, string caBoneName, const vector& tPositionOffset, const vector&
tAngleOffset, bool bLooped, enum MonsterSound::EType sound_type);
    void sound (sound_object& sound, const vector& tPosition, const vector& tAngleOffset, bool bLooped,
enum MonsterSound::EType sound_type);
//конструкторы
    void sound ();
//конструкторы для сталкеров и монстров
    void sound (string caSoundToPlay, string caBoneName);
    void sound (string caSoundToPlay, string caBoneName, const vector& tPositionOffset);
    void sound (string caSoundToPlay, string caBoneName, const vector& tPositionOffset, const vector&
tAngleOffset);
    void sound (string caSoundToPlay, string caBoneName, const vector& tPositionOffset, const vector&
tAngleOffset, bool bLooped);
        //<sound_name>, <bone_name>, <position_offset>, <angle_offset>, <looped>. 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. Первые два параметра
обязательны, остальные по умолчанию равны нулю
    void sound (string caSoundToPlay, const vector& tPosition);
    void sound (string caSoundToPlay, const vector& tPosition, const vector& tAngleOffset);
    void sound (string caSoundToPlay, const vector& tPosition, const vector& tAngleOffset, bool bLooped);
        //см. выше, играется по умолчанию от головы
    void sound (sound_object& sound, string caBoneName, const vector& tPositionOffset);
    void sound (sound_object& sound, string caBoneName, const vector& tPositionOffset, const vector&
tAngleOffset);
    void sound (sound_object& sound, string caBoneName, const vector& tPositionOffset, const vector&
tAngleOffset, bool bLooped);
        //см. выше, инициализуется не строковым именем, а объектом звука (см. класс sound_object)
    void sound (sound_object& sound, const vector& tPosition);
    void sound (sound_object& sound, const vector& tPosition, const vector& tAngleOffset);
    void sound (sound_object& sound, const vector& tPosition, const vector& tAngleOffset, bool bLooped);
        //см. выше, инициализуется не строковым именем, а объектом звука (см. класс sound_object),
играется по умолчанию от головы
//конструкторы для монстров
    void sound (enum MonsterSound::EType sound_type);
    void sound (enum MonsterSound::EType sound_type, int delay);
        //sound_type — тип звука (см. перечисление выше), delay — задержка звука.
```

```
//этот конструктор предполагает, что объект еще будет корчить рожи. Но, поскольку это умеет только Сидор,
видимо, это для торговцев (CAI_Trader). У меня сталкеры выражение лица не меняли.
    void sound (string caSoundToPlay, string caBoneName, enum MonsterSpace::EMonsterHeadAnimType
head_anim_type);
        //caSoundToPlay – имя звука, caBoneName – имя кости, head_anim_type - член перечисления
MonsterSpace, анимация лица.

    bool completed();
        //закончен ли экшн
    void set_position(const vector& pos);
        //установить координаты звука
    void set_bone(string caBoneName);
        //установить кость, от которой играется звук
    void set_angles(const vector& angle);
        //установить углы (?) звука
    void set_sound(string caSoundToPlay);
        //установить звук
    void set_sound(const sound_object& sound);
        //установить звук
    void set_sound_type(enum ESoundTypes sound_type);
        //установить тип звука. Тип брать из перечисления snd_type.
};

class sound_object {                            //параметры отыгрывания звуков
    const looped = 1;
    const s2d = 2;
    const s3d = 0;

    property frequency;
    property max_distance;
    property min_distance;
    property volume;

    sound_object (string);
    sound_object (string, enum ESoundTypes);

    function set_position(const vector&);
    function stop_deffered();
    function get_position() const;
    function play_no_feedback(game_object*, number, number, vector, number);
    function play_at_pos(game_object*, const vector&);
    function play_at_pos(game_object*, const vector&, number);
    function play_at_pos(game_object*, const vector&, number, number);
    function stop();
    function length();
    function play(game_object*);
    function play(game_object*, number);
    function play(game_object*, number, number);
    function playing() const;
};
```

В целом - все понятно. Создаем объект действия одним из конструкторов, недостающие данные заполняем методами класса. После добавления действия в состояние его нельзя ни удалить, ни изменить.

Рабочий пример:

```
function execute_act(cObj)
    cObj:script(true, "my")  --cObj - клиентский объект какого-нибудь монстра
    add_action(cObj, act(act.rest), cond(cond.time_end, 20000))
    add_action(cObj, act(act.attack, db.actor), cond(cond.time_end, 20000))
    add_action(cObj, act(act.panic, db.actor), cond(cond.time_end, 20000))
end

function add_action(cObj, ...)
    local arg = {...}
```

```
    local act = entity_action()
    local i = 1
    while true do
        if (arg[i] ~= nil) then
            act:set_action(arg[i])
        else
            break
        end
        i = i + 1
    end
    if (cObj ~= nil) then
        cObj:command(act,false)
    end
    return  entity_action(act)
end
```

После запуска этого скрипта свинка пойдет под ближайшее дерево отдыхать, через 200 секунд встанет и нападет на актора, еще через 200 секунд будет от актора убегать в панике.

Теперь об ограничениях. В движке управление монстрами и сталкерами реализовано так: планировщик поведения монстров построен на концепции FSM (Finite State Machine), сталкерский планировщик - на концепции GOAP (Goal-Oriented Action Planning). Хотя в билдах сталкерами тоже можно было полностью рулить на FSM, к финалке это было урезано, так что для сталкеров некоторые действия не работают - это move и look. Возможно, я ошибаюсь, если у кого-нибудь получится заставить ходить сталкера под FSM, дайте знать.

Теперь, стало быть, разберем, как рулить сталкерами с помощью GOAP. Начальное слово, опять же, предоставлю Ясеневу :)

"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."

То есть суть такова: имеем планировщик, привязанный к объекту. Планировщик постоянно вызывает эвалуаторы (спецфункции, проверяющие, выполнены ли предусловия для действия). Если эвалуатор возвращает true, планировщик запускает действие.

Автор: К.Д.

Источник — «https://xray-engine.org/index.php?title=Управление_персонажем&oldid=850»

Категории:

A-Life
Недописанное