INTRODUCCIÓN A LA PROGRAMACIÓN DE VIDEOJUEGOS (IPV)

INTELIGENCIA ARTIFICIAL

Depto. Sistemas Informáticos y
Computación. UPV

Objetivos de aprendizaje

Introducir a los conceptos básicos de la I.A. aplicada a los videojuegos

Aplicar los conocimientos de máquinas de estados finitos a la gestión del comportamiento de los personajes sintéticos en los videojuegos

Almacenar y gestionar la información en el ambiente que rodea a los personajes del juegos como ayuda a la toma de decisiones

Índice

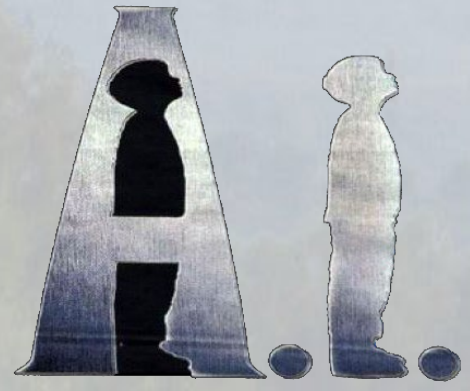
Introducción (I)

Inteligencia. Definición

- Inteligere. Compuesto de intus "entre" y legere "escoger"
- Etimológicamente, inteligente es quien sabe escoger
- La inteligencia elige la mejor opción para alcanzar un objetivo
- Introducida por Cicerón. Determina la capacidad intelectual

Inteligencia Artificial. Definición

- Rama de la informática
- Estudia los procesos de elección de opciones para alcanzar un objetivo
- Elecciones basadas en
  - La secuencia de entradas percibidas
  - El conocimiento almacenado



Introducción (III)

Inteligencia Artificial en **videojuegos**. Definición

- Cualquier comportamiento en una entidad de juego tendente a conseguir un objetivo
- No importa técnica empleada
- IA académica = IA en videojuegos
- IA en videojuegos ≠ IA académica
  - Una simple condición if-then-else
  - AFD
  - Localizador de caminos
  - Generación automática de entornos

Introducción (IV)

Inteligencia Artificial en videojuegos

Muy limitada por restricciones de tº real

Configurable por diseñadores y/o jugadores

Nunca debe retrasar lanzamiento videojuego

No existe solución genérica para cualquier videojuego

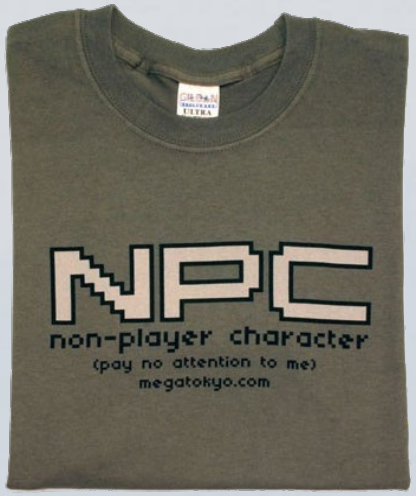
- Estrategia: análisis campo batalla // Planificación a largo plazo
- FPS: Análisis táctico. Movimientos inmediatos
- RTS: Los que más IA incluyen

Personajes pueden actuar como oponentes, aliados o neutrales

Definiciones

NPC. Non Player Character

- Personaje que no representa a un jugador humano
- **Simulación** controlada por el **ordenador**
- Puede tener representación humanoide o no
- En RPG, controlado por el gamemaster (GM)



Avatar

- Encarnación terrestre de un dios hindú (Vishnú)
- **Representación** virtual de un **jugador**

Scripting (I)

Permite incorporar nuevo contenido al videojuego sin necesidad de dedicar más programadores

Sistema simplificado de descripción de IA, lógica o comportamientos

Lenguajes empleados: desde lenguajes de marcado hasta alto nivel

Selección de lenguaje a emplear depende de funcionalidad en ejecución del lenguaje, tamaño de los scripts y perfil de potenciales usuarios

Existen lenguajes de script bastante asentados:

LUA. Pequeño, rápido, fácil de aprender e integrar en C++

Python,...

GameMonkey,...

Unrealscript, Action Script,...

Scripting (II)

Facilitan el prototipado rápido, la accesibilidad, sintonización del código o el escalado de los programas

Los personajes ya no calculan respuesta sino que siguen secuencia de comandos del script

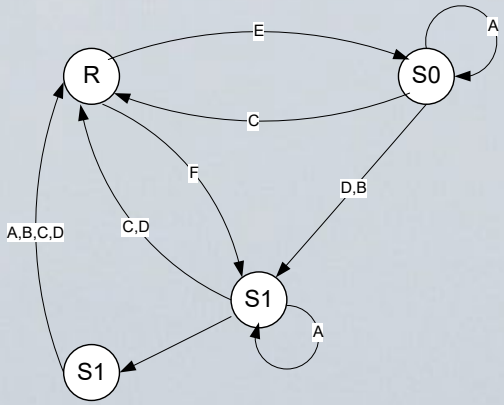
Inconvenientes

- Velocidad de ejecución interpretada. Más lenta que código ad-hoc específico dentro del programa
- Dificultad de depuración mayor
- Dificultad para balancear expresividad y potencia del lenguaje en función de casos de uso

FSM (I)

Una FSM es una entidad abstracta compuesta por

- Un conjunto de **estados**
- Un **estado inicial**
- **Transiciones** que relacionan las entradas, los estados actuales y los destinos
- Un **vocabulario** de **entrada**
- **Acciones** a realizar en cada estado y transitorio



FSM (II)

Muy utilizadas en videojuegos

Los juegos comerciales utilizan FSM en todo momento (Los Sims)

Pueden combinarse con otros métodos

Pueden llegar a dar sensación de comportamiento inteligente, pero predecible. Sol: Emplear Non-Deterministic Autómata (NDA)

Implementados con programación ad-hoc

Correspondencia natural entre estados y comportamientos

Fácilmente esquematizable, programable y depurable

Solución general a cualquier tipo de comportamiento

FSM (III)

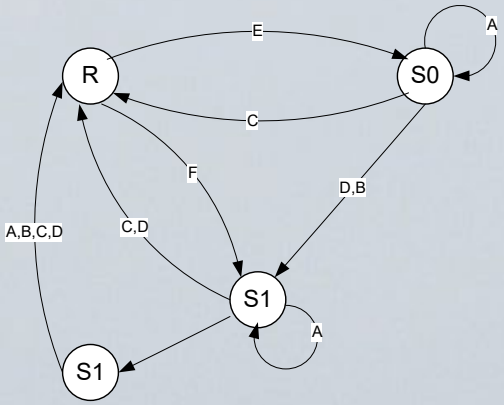
Estados

Una FSM puede tener tantos estados como sea necesario. Cuidado: explosión estados si comportamiento es complejo

Sólo puede estar en un estado la vez

Debe definirse siempre un **estado inicial** de arranque

Pueden existir estados transitorios



Transiciones

Llevan a la FSM de un estado a otro

Para que ocurra una transición, deben cumplirse una serie de condiciones. Diferentes o no, según el estado en el que se encuentre

Las condiciones pueden ser todo lo complejas que sea necesario

FSM (IV)

Acciones

Descripción de una actividad que la FSM puede ejecutar en un estado concreto (n)

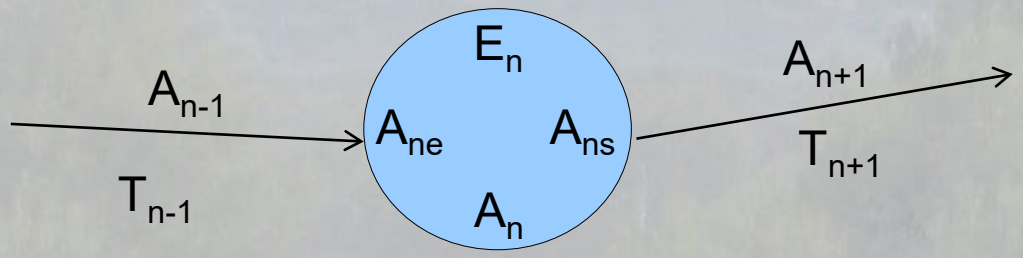
Se dividen en 4 tipos:

Entrada. Acciones que se ejecutan al entrar a un estado. Inicializaciones. A_{ne}

Salida. Acciones que se ejecutan al salir de un estado. Liberación de recursos. A_{ns}

Estado. Acciones que se ejecutan regularmente por estar en un estado. A_n

Transición. Acciones que se ejecutan después de la salida de un estado y antes de que se ejecuten las acciones de entrada al siguiente estado. A_{n-1} y A_{n+1}



FSM (V)

Minero en la mina

Estados

Buscando (B), Descansando (D) y Comiendo (C)

Inicial: Buscando

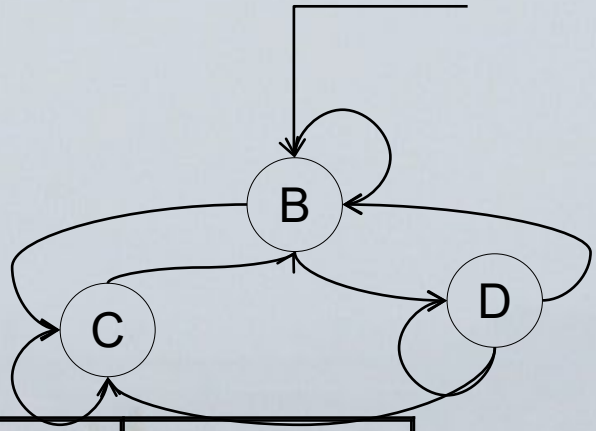
Vocabulario entrada / condiciones de disparo

Cansado (C)

Descansado (D)

Hambriento (H)

Saciado (S)



Evnt/Estd	*Buscando*	*Descansa*	*Comiendo*
C	**Descansa**	Descansa	Comiendo
D	Buscando	**Buscando**	Comiendo
H	**Comiendo**	**Comiendo**	Comiendo
S	Buscando	Descansa	**Buscando**

FSM (VI)

Implementación

- ✓ **Lenguaje imperativo**: sucesión de sentencias *if* o *case*
- ✓ **Lenguaje lógico** de IA
- ✓ Facilita la programación
- ✓ No disminuirá la complejidad del problema

```
switch (estado)
    {
    case nombre_estado:
       [Acciones por defecto]
       [Evaluación de Condición]
       if (transición)
             estado=estado_destino;
       (...)
       break;
    (…)
    }
```

FSM (VIIa)

Lenguaje imperativo

¿Dónde se ejecutan la acciones de estado A_n?

¿Dónde se ejecutan la acciones de salida A_{ns}?

```
void RunLogic( int * state ) {
    switch( *state )
    {
        case WANDER:
            Wander();
            if( SeeEnemy() ) { *state = ATTACK; }
            break;
        case ATTACK:
            Attack();
            if( LowOnHealth() ) { *state = FLEE; }
            if( NoEnemy() ) { *state = WANDER; }
            break;
        case FLEE:
            Flee();
            if( NoEnemy() ) {
              ResetSpeed();
              *state = WANDER; }
            break;
    }
}
```

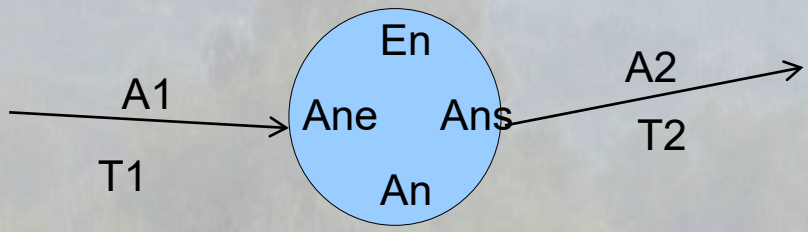
A_n A_n A_n A_{ns} A_{ns} A_{ns} A_{ns}

El lenguaje no apoya a la estructura del problema

Las transiciones entre estados se realizan mediante polling

No puede editarlo ni definirlo el jugador o los diseñadores (ejecución)

Empleo de macros ≈ scripting



FSM (VIIb)

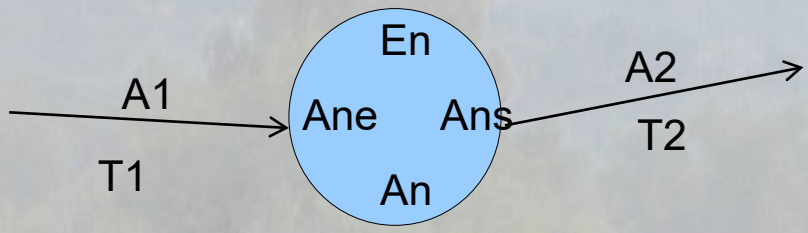
```
void RunLogic( int * state ) {

        switch( *state )
        {
            case WANDER:
                Wander();
                if( SeeEnemy() ) { *state = ATTACK; }
                break;
            case ATTACK:
                Attack();
                if( LowOnHealth() ) { *state = FLEE; }
                if( NoEnemy() ) { *state = WANDER; }
                break;
            case FLEE:
                Flee();
                if( NoEnemy() ) {
                  Acc = -0,5;
                  *state = BREAKING; }
                Break;
            case BREAKING:
                Speed += Acc;
                If( 0,0 >= Speed ) {
                  Acc = Speed = 0,0;
                  *state = WANDER; }
                break;
        }
    }
```

¿Dónde se ejecutan la acciones de Transición de entrada A1 y de salida A2?

Introduciendo nuevos estados transitorios

Para pasar de *Flee* a *Wander* hay que ir frenando hasta que el personaje se para y puede hacer *Wander*



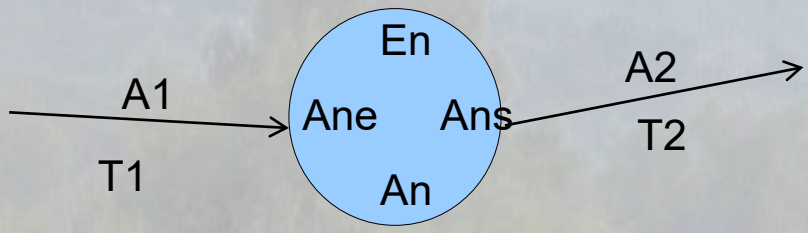
FSM (VIIc)

```
void RunLogic( int * state ) {
        static bool semaphore = true;
    switch( *state )
    {
        case WANDER
            if (semaphore){
                PutLeftHandOnEyes();
                semaphore = false;
            }
            Wander();
            if( SeeEnemy() ) { *state = ATTACK; }
            break;
        case ATTACK:
            Attack();
            if( LowOnHealth() ) { *state = FLEE; }
            if( NoEnemy() ) { *state = WANDER; }
            break;
        case FLEE:
            Flee();
            if( NoEnemy() ) {
              ResetSpeed();
              *state = WANDER; }
            break;
    }
}
```

¿Dónde se ejecutan la acciones de entrada A_{ne}?

Empleando condición de entrada con semáforo

La primera vez que se hace WANDER, hay que subir la mano a los ojos



FSM (VIII)

Lenguaje de script

```
AgentFSM
{
        State( STATE_Wander )
                OnUpdate
                   Execute( Wander )
                   if( SeeEnemy ) SetState( STATE_Attack )
                OnEvent( AttackedByEnemy )
                   SetState( Attack )
        State( STATE_Attack )
                OnEnter
                   Execute( PrepareWeapon )
                OnUpdate
                    Execute( Attack )
                   if( LowOnHealth ) SetState( STATE_Flee )
                   if( NoEnemy ) SetState( STATE_Wander )
                OnExit
                   Execute( StoreWeapon )
        State( STATE_Flee )
                OnUpdate
                    Execute( Flee )
                   if( NoEnemy ) SetState( STATE_Wander )
}
```

- El lenguaje apoya la estructura del problema
- Permite eventos y polling
- Existen los conceptos OnEnter y OnExit
- Puede editarlo y definirlo el jugador o los diseñadores
- Curva aprendizaje menor que C/C++
- Menos expresividad o comportamiento más limitado
- Se pueden usar híbridos entre los dos métodos

FSM (IX)

Minero en la mina
Cada uno de los estados puede explotarse en otra FSM

Estado B (Buscando)

Estados

De Pie (DP), Encender linterna (EL), Apagar Linterna (AL), Colocarse el casco (CC), Quitar el casco (QC), Tomar pico (TPi), Picar (Pi) y Dejar Pico (DPi), Tomar Pala (Tpa), Dejar Pala (Dpa), Palear (Pa)

Inicial: De pie (DP)

Vocabulario entrada / condiciones de disparo

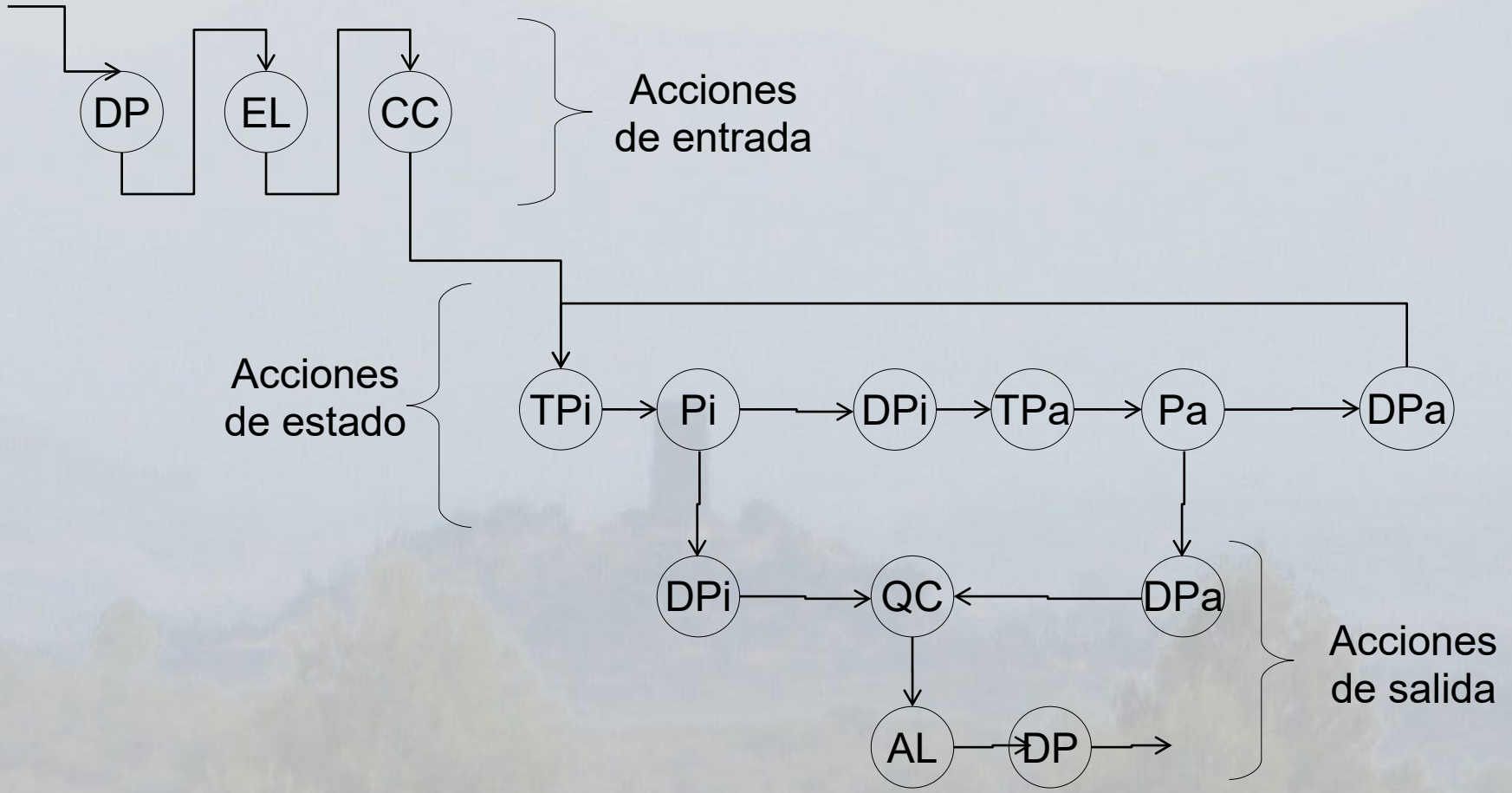
Desescombrar (D)

Limpio (L)

Finalizar trabajo (F)

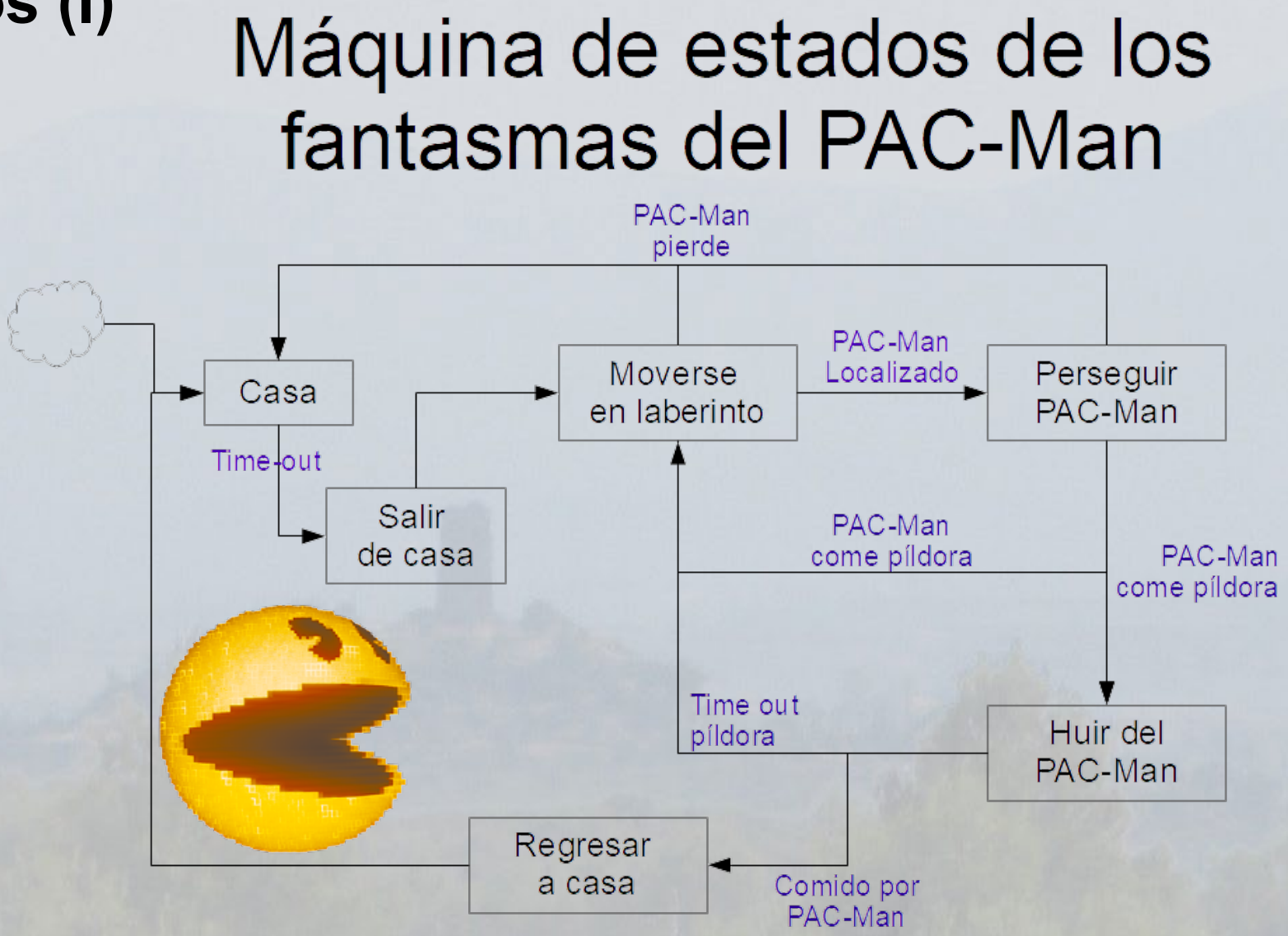
FSM (X)

Minero en la mina (Buscando)



FSM (XI)

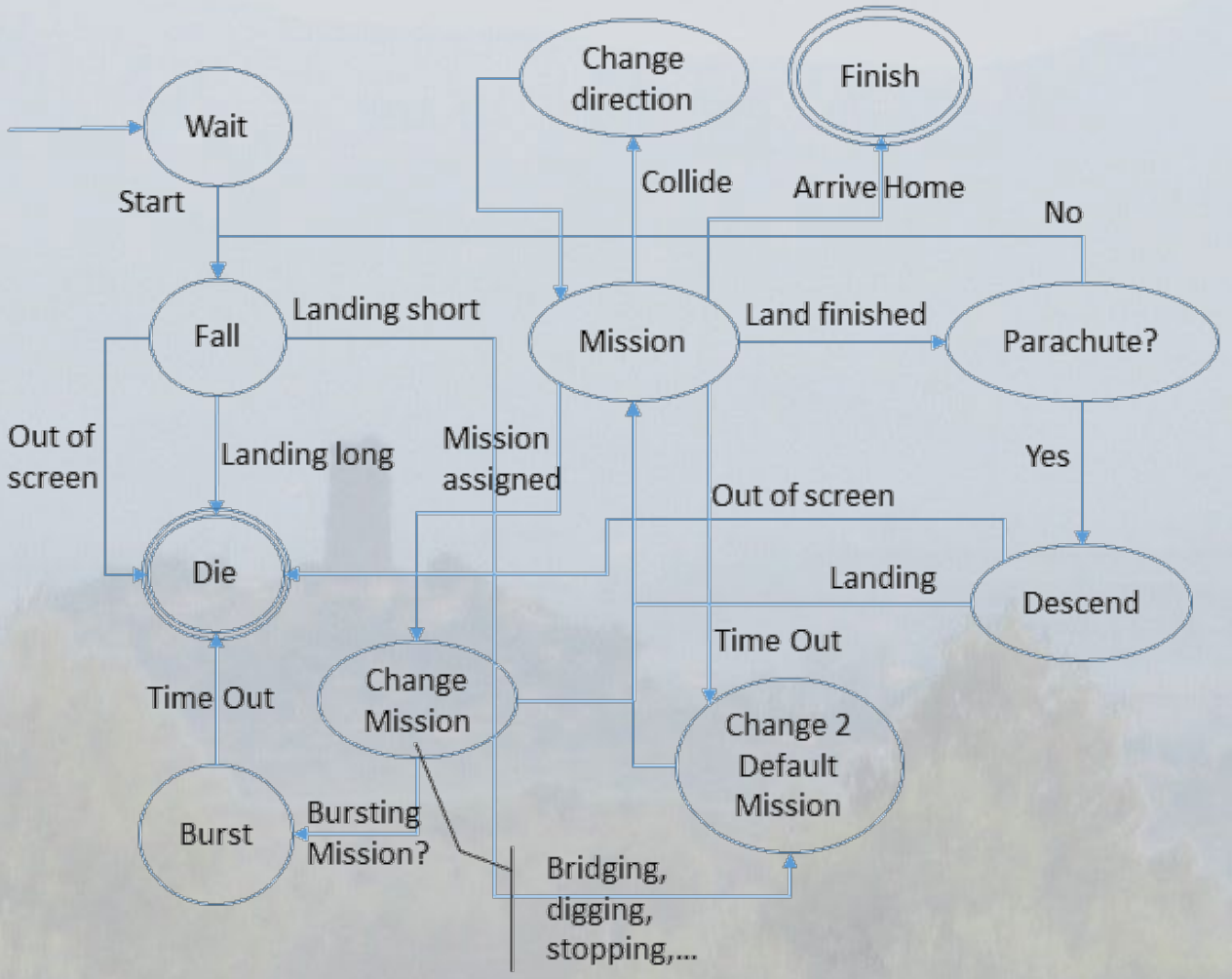
Ejemplos (I)

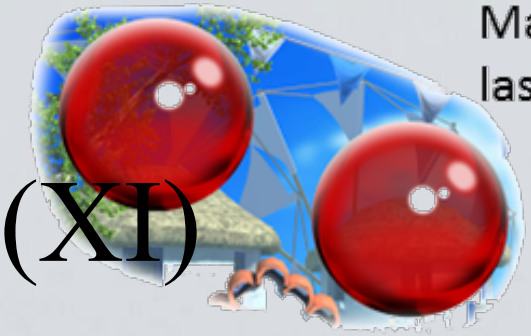


FSM (XI)

Ejemplos (II)

Máquina de estados de los lemmings

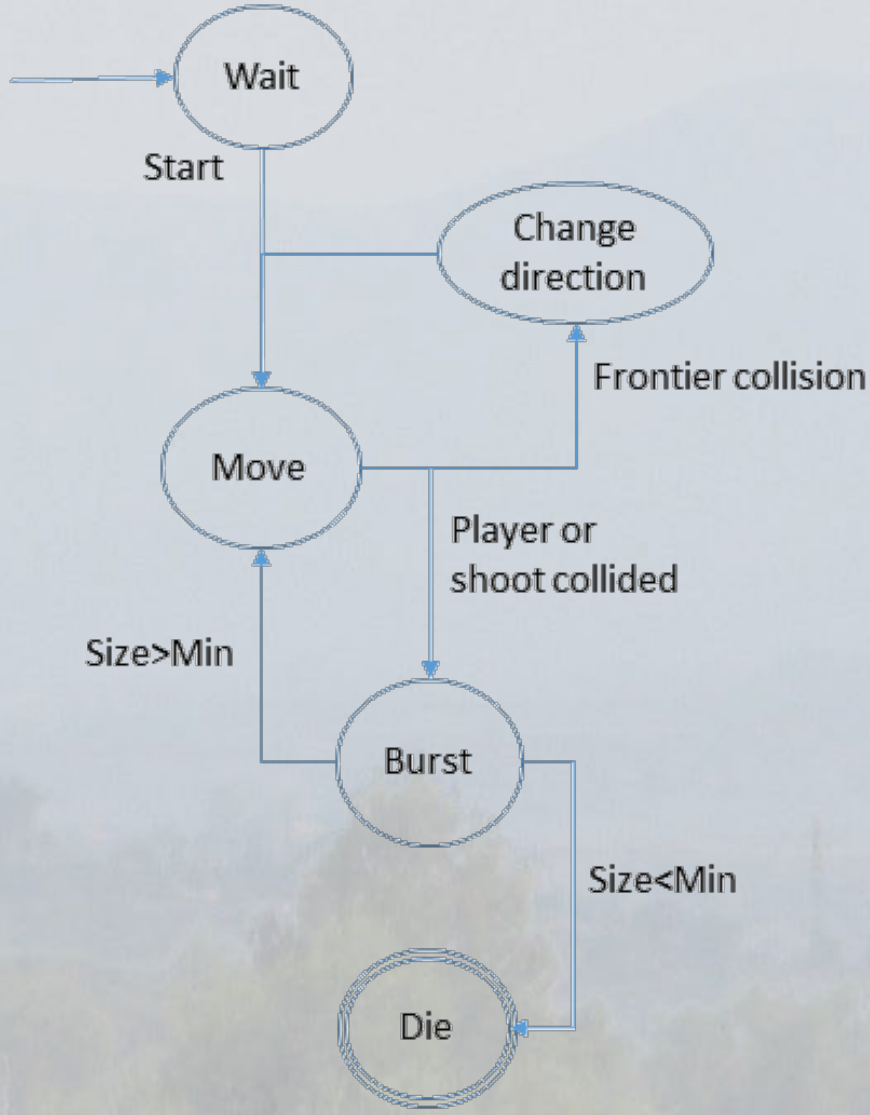




Máquina de estados de las burbujas de Pang

FSM (XI)

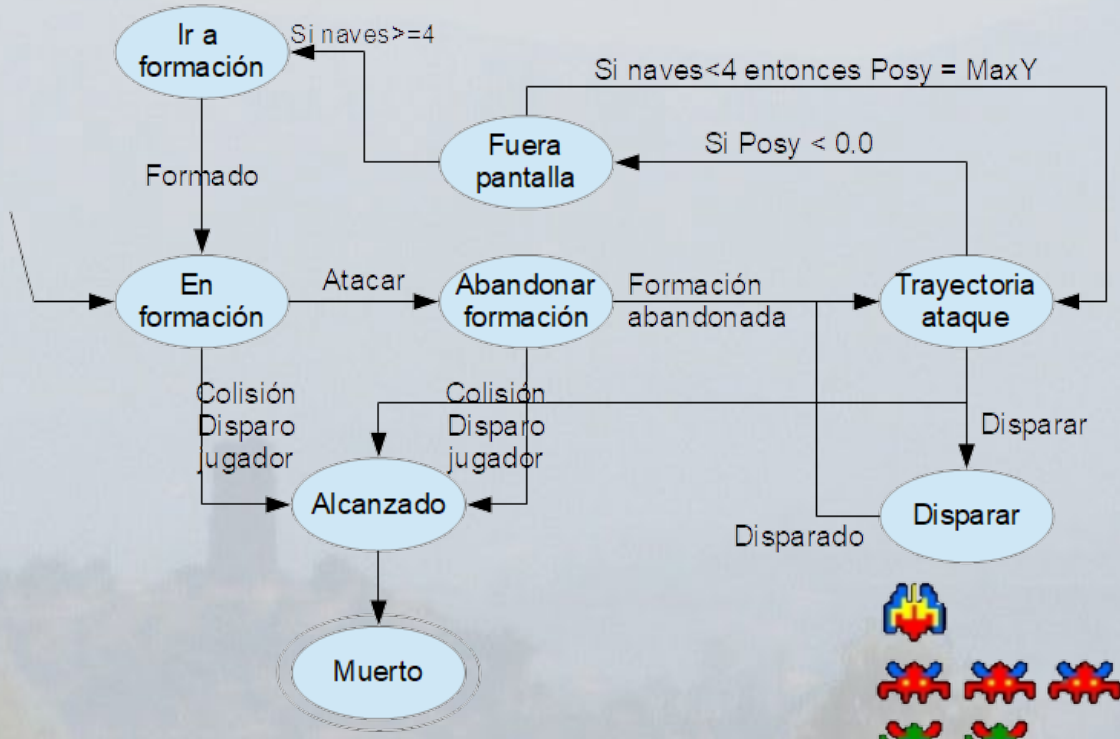
Ejemplos (III)

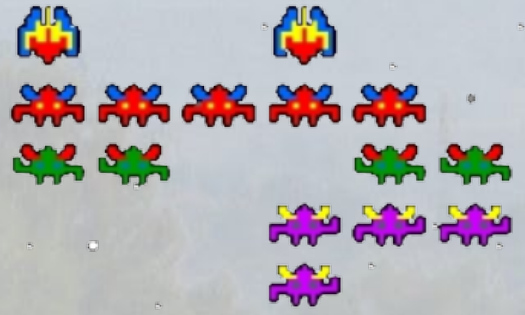


FSM (XI)

Ejemplos (IV)

Diagrama de estado-transición naves videojuego Galaxians





FSM (XII)

Ventajas

Útiles para modelar comportamientos sencillos y repetitivos

Fáciles de programar y depurar

Gran rapidez (sencillez)

Intuitivas, fáciles de comprender

Flexibles, se pueden expandir fácilmente

Para IA compleja, emplear otras técnicas

Decisiones resultados de IA compleja pueden ser implementadas por FSM.

Ejemplos:

Red neuronal responde que hay que desplazarse hasta punto A

Desplazamiento hasta punto A realizado por FSM que implementa acción y animación de caminar

Animaciones predefinidas siempre de realizan por FSM

FSM (XIII)

Inconvenientes

- Basadas en reglas simples
- Suelen presentar comportamiento predecible (AFD)
- Disimulable por aleatoriedad (AFND)
- Crece el total de estados con la complejidad del problema
- No es apropiada para: Resolver rompecabezas, decidir armamento para atacar, seleccionar la mejor ruta, camino entre dos puntos con obstáculos,...
- Emplear otras técnicas de IA para: construir planes, resolver problemas y crear tácticas

FSM (XIV). Extensiones (I)

Anidación de FSM / FSM jerárquicas

Si complejidad del sistema aumenta, es difícil representarlo mediante una única FSM

Emplear metodología Top-Down

Romper las FSM en sub-FSM internas más simples

Cada sub-FSM debe intercambiar el mínimo de información posible con el resto de sub-FSM

Permite diseños más modulares e independientes

Ejemplo

Unreal 2 hay una FSM general para controlar a cada enemigo

Dentro de cada estado existen otras FSM para realizar tareas específicas:

Defender una zona de ataques enemigos

Explorar el entorno de juego

FSM (XV). Extensiones (II)

FSM basadas en mensajes o eventos (I)

En algunos juegos o algunos estados, determinadas transiciones se invocan muy raramente.

Si

- La resolución de esta transición es computacionalmente costosa
- Se está empleando un modelo de polling para determinar cuando se realiza ésta

Entonces se puede disparar transición al nuevo estado

- Detectando la superación de umbrales de atributos internos
- Mediante el envío de un mensaje en lugar de realizar polling permanentemente

FSM (XVI). Extensiones (III)

FSM basadas en mensajes o eventos (II)

Necesario implementar método adicional en el FSM que cambie al estado correspondiente cuando se recibe un evento

Se puede enriquecer estado indicando si se activa por evento o por transición (polling)

FSM crea listas de estados a activar a medida que se los carga en la fase de inicialización

FSM (XVII). Extensiones (IV)

FSM borrosas (I)

Fuzzy State Machines, FuSM

FuSM ≠ lógica difusa obligatoriamente

FuSM = Lógica convencional extendida para soportar verdades parciales

FuSM = FSM que puede estar simultáneamente en varios estados

FuSM emplea nivel de activación para determinar probabilidad de estar en un estado determinado

FSM (XVIII). Extensiones (V)

FSM borrosas (II)

Tipos

Con transiciones priorizadas

- En cada iteración de su IA, se evalúan TODOS los estados del FSM
- Aquel estado que tenga el mayor nivel de activación pasa a ser el estado actual

Aún así, el sistema sigue siendo una FSM ordinaria

Transiciones probabilistas
Cada transición, incluso cuando se activa, tiene una probabilidad de ejecutarse

La probabilidad puede ser dinámica o fijada en la fase de inicialización

FSM (XIX). Extensiones (VI)

FSM borrosas (III)

Tipos

Modelos de Markov

Son transiciones totalmente probabilistas

Ej: El 98% del tiempo el coche arrancará y sólo un 2% fallará en el arranque

Función de confianza que determina porcentaje de veces que ocurrirá una determinada transición.

Útiles cuando

- Hay que determinar toma de riesgos
- No hay suficientes indicios para decidir qué transición tomar
- Simular tasas de fallos o tolerancias, comportamientos imperfectos o impredecibles,...

FSM (XX). Extensiones (VII)

FSM borrosas (IV)

Tipos

Lógica difusa real

- Emplea reglas expresadas en verdades parciales
- Se emplean matrices de reglas que deben combinarse

Problema principal:

- Explosión de estados si existen muchas variables a tener en cuenta
- Dificultad en modelar reglas ambiguas

FSM (XXI). Extensiones (VIII)

FSM basadas en pilas

El estado del FSM se sustituye por una pila de estados

Permite la gestión de estados globales o prioritarios. Estados que

- Son capaces de **interrumpir los demás**
- Deben ejecutarse inmediatamente
- Cuando se acaba su gestión, el FSM debe volver al estado anterior interrumpido

Por lo tanto, es necesario una máquina de pilas que **recuerde el estado interrumpido** para poder regresar a él tras la finalización de la ejecución del estado global.

Ejemplo:

Estado global del minero: **Ir al baño**

Estando en cualquier estado, cuando se cumpla la condición **vejiga llena,** dejará lo que esté haciendo e irá al baño de inmediato

Después, volverá a lo que estaba haciendo: trabajar, comer,...

FSM (XXII). Extensiones (IX)

FSM con inercia

Problema que suele aparecer en FSM es la oscilación de estado
Forma de actuar de personajes que no estaba explícitamente programada

Surge como consecuencia de la interacción de reglas simples del autómata

Ej: Si jugador de fútbol tiene línea de visión directa a la portería, dirigirse hacia él y bloquear línea.

Si línea se activa o desactiva frecuentemente, jugador contrario realiza muchas oscilaciones en poco tiempo entre estado de pie y estado corriendo

Solución

Incluir inercia. Disminuir la tendencia al cambio:

- Obligando a estar en ese mismo estado un mínimo de tiempo o a no cambiar hasta que se hayan percibido varias veces la misma transición
- Dando más peso a las percepciones más antiguas y menos a las modernas. Suma ponderada
- Dar más prioridad a la tendencia que al evento

FSM (XXIII). Extensiones (X)

FSM concurrentes

Dos escenarios:

Un mismo FSM compartido entre todos los personajes que muestran el mismo comportamiento. Existe un AI manager que determina comportamiento dependiendo del estado particular del personaje. Ej: Varios personajes corriendo en poses diferentes

Diferentes comportamientos (FSM) controlando al mismo personaje. Ej:

- FSM para secuencia de animación de carrera junto con otro de IA y un tercero de disparo
- Ejército enemigo dispone de FSM para determinar gestión de recursos, táctica de combate,...

Necesario super FSM que coordine vía mensajes o zona de memoria compartida

Cuidado con comportamientos emergentes

Información basada en la posición(I)

Emplean tres técnicas fundamentalmente

- Mapas de Influencia
- Terreno inteligente
- Análisis del terreno

Empleados en búsqueda de caminos sobre terreno, movimientos de fichas,...

Información basada en la posición(II)

Mapas de Influencia (MI)

Rejilla superpuesta al terreno. Vector o matriz bidimensional con descripción de contenidos en cada celda ≈ GIS

Problema

- Resolución dependiente de compromiso entre precisión y memoria disponible

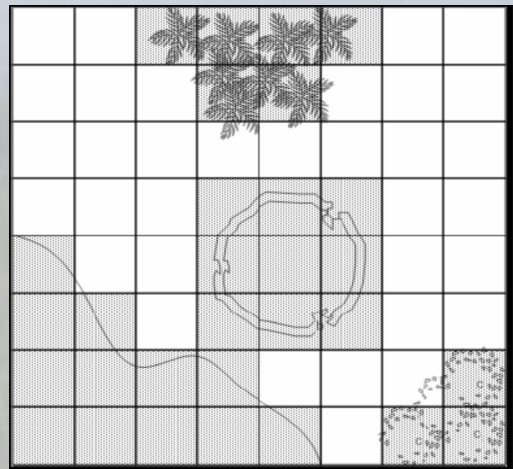
Solución

Emplear LoD MI ≈ Google Maps. Acceder al nivel de detalle según personaje de juego o posición

MI locales a la zona de actividad (batalla)

Facilita identificación de

- Frente de batalla
- Áreas no vigiladas
- Selección de camino óptimo



Información basada en la posición(III)

Mapas de Influencia (MI)

Emplear capas de terreno. Sim City

Atributos de cada celda:

- Tipo de terreno: agua, pantanoso, árboles, gravilla, duro,...
- Estado: minado, lucha, vigilado,....
- Un objeto ocupa una o más celdas o una celda puede estar ocupada / afectada por varios personajes

Información basada en la posición(IV)

Terreno inteligente (TI)

Empleado por Will Wright en Los Sim

En cada celda:

- Lógica y comportamiento dentro del objeto
- Objetos responden a necesidades de personajes. Ej: cama responde a cansancio NPC o microondas a hambre
- Objetos indican a NPC qué necesidades satisfacen
- NPC actúa en consecuencia

Información basada en la posición(V)

Análisis del Terreno (AT)

Conjunto de métodos que permiten obtener conocimiento al sistema de IA acerca del terreno

Especialmente útil cuando

- Los terrenos son generados aleatoriamente al vuelo. No existe preproceso en fábrica en fase de diseño
- Búsqueda de objetivos útiles táctica o estratégicamente
- Incluso con LoD MI, algoritmos de reconocimiento de patrones pueden ser muy costosos

Información basada en la posición(VI)

Casos de uso

Determinación de ocupación

Estimaciones de tamaño de tropas, densidad de población,... Caso típico Niebla de guerra: ámbito de visión en RTS. Obliga a explorar mapa para encontrar recursos, enemigos,...

Control de terreno

Asignación de terreno a cada bando dependiendo de tamaño, localización y armamento que posea

Localización de caminos

Determinación de trayectorias a seguir por los personajes para evitar caer en terrenos enemigos, hostiles, intransitables,...

Determinación de peligros temporales

Determinación de localizaciones en las que ha estado ocurriendo peligros durante un determinado tiempo

Bibliografía

Cap 5.3. Introduction to Game Development. Steve Rabin. Charles River Media ISBN: 978-1-58450-377-4

Artificial Intelligence, A Modern Approach, Stuart J. Russell and Peter Norvig, Prentice Hall, ISBN 0-13-103805-2

AI Game Engine Programming, Brian Schwab, Charles River Media, Inc, 2004 ISBN: 1-58450-344-0

creative commons
COMMONS DEED

Documentación generada por
Dr. Ramón Mollá Vayá
Sección de Informática Gráfica
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia