INTELIGENCIA ARTIFICIAL EN VIDEOJUEGOS

Andrés Pastorini - Alvaro Martínez

TDIVJ2D – Tecnólogo Informático

Inteligencia Artificial en Videojuegos

TEMARIO

1. **Introducción**
2. Chasing and evading
3. Patrones de movimiento
4. Focking
5. Busqueda de caminos
6. Máquinas de estado finito
7. Fuzzy logic
8. IA basada en reglas

1. Introducción

Inteligencia Artificial (para videojuegos):

Todo lo que de la ilusión de inteligencia en un nivel adecuado que haga al juego más estimulante y sobre todo divertido puede ser considerado IA.

1. Introducción

Determinista

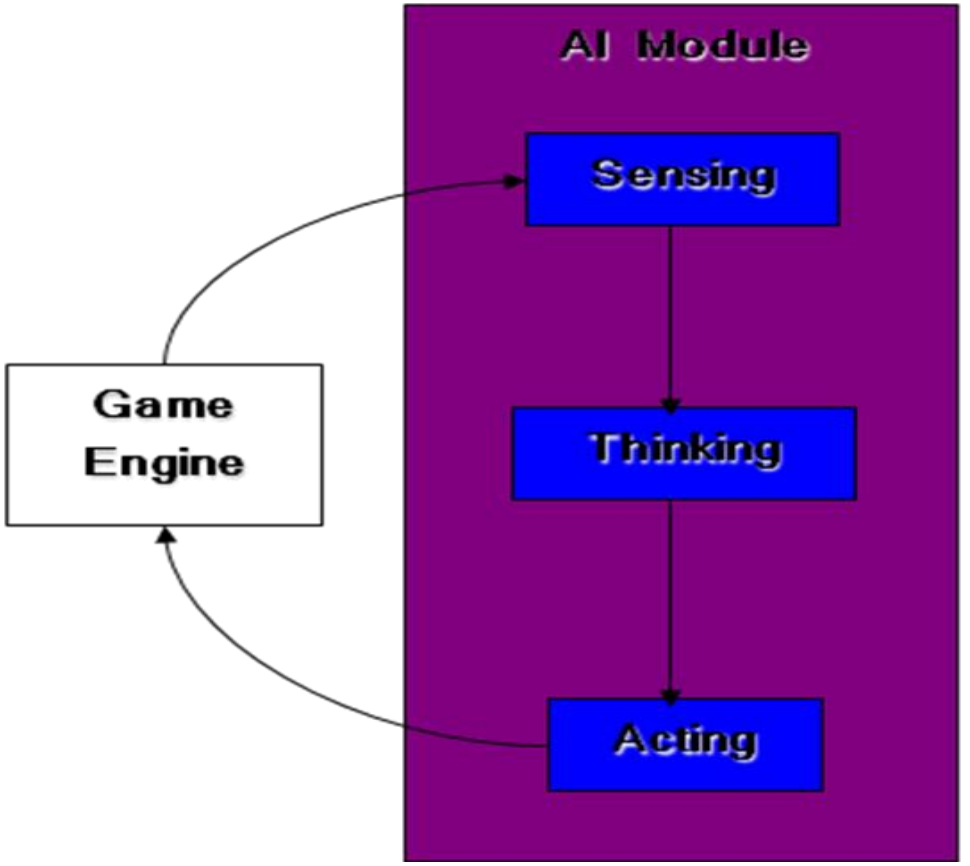
Es específico y predecible. No hay incertidumbre. Por ejemplo es un algoritmo de persecución simple donde un NPC mueve hasta llegar a las coordenadas del objetivo.

No deterministas

Conducta que tiene un grado de incertidumbre y es algo impredecible (el grado de incertidumbre depende del método de IA empleado). Un ejemplo de comportamiento no determinista es un NPC aprendiendo para adaptarse a las tácticas de lucha de un jugador. El aprendizaje podría lograrse usando una red neuronal, una técnica Bayesiana, o un algoritmo genético.

1. Introducción

Se actualiza dentro del game loop, luego de procesar los eventos de usuario y antes de renderizar.



No tiene porque actualizar en cada frame, ni es necesario que todos los agentes se actualicen en todos los frames.

La programación de ia puede llevar el 50% del tiempo de construcción!!!

1. Introducción

Distintas formas de implementar AI

Polling
El sistema de IA es convocado cada cierto periodo fijo de tiempo. Censa el mundo para conocer su estado actual, que elementos puede ver, si hay animaciones ejecutándose, etc.
Luego de conocer el estado del mundo actúa

Distintos npc pueden requerir distinta frecuencia de muestreo.

1. Introducción

Distintas formas de implementar AI

Orientada a eventos
El modulo de IA actúa al recibir un evento del mundo(un mensaje con información que es procesado por determinada función).

Ejemplos de disparadores de eventos puede ser el paso de determinada cantidad de tiempo,
el que algo haya entrado al campo de visión de un personaje,etc.

Los mensajes no pueden suplantar completamente la fase de censo, al recibir un evento puede ser necesario censar.

Inteligencia Artificial en Videojuegos

TEMARIO

1. Introducción
2. **Chasing and evading**
3. Patrones de movimiento
4. Focking
5. Búsqueda de caminos
6. Máquinas de estado finito
7. Fuzzy logic
8. IA basada en reglas

2. Chasing and Evading

Sin obstáculos

En cada iteración del game loop la ia busca que la distancia entre el cazador y la presa disminuya.

Si se utiliza física, para cada posición y velocidad en determinado momento, se obtiene la fuerza(impulso, dirección) para el nuevo movimiento.

Si el entorno es continuo cada posición puede ser cualquier número real, en cambio si es un juego basado en tiles, la posición del jugador esta asociado a una grilla, por lo que los movimientos serán valores discretos.

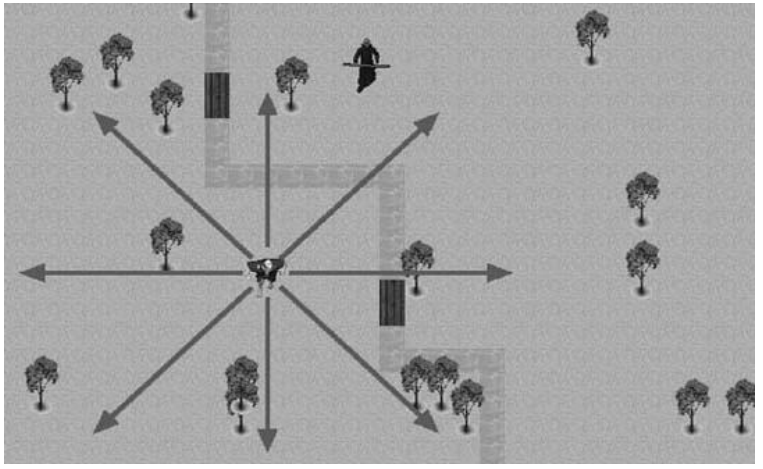
2. Chasing and Evading

Sin obstáculos

En “tiled enviroment”, el escenario es dividido en un número discreto de tiles.

El movimiento en estos escenarios es más restrictivo.

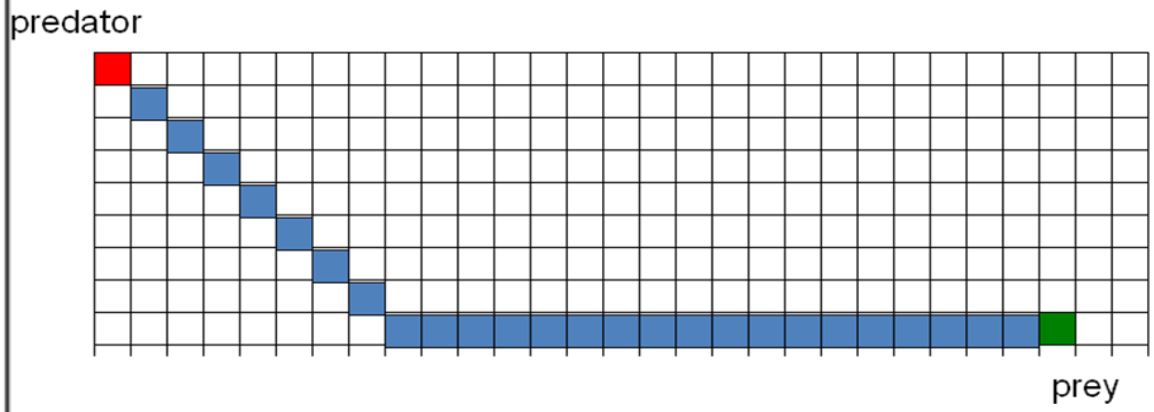
Para evitar movimiento jaggy (dentado) se debe mover en tiles adyacentes.



2. Chasing and Evading

Sin obstáculos

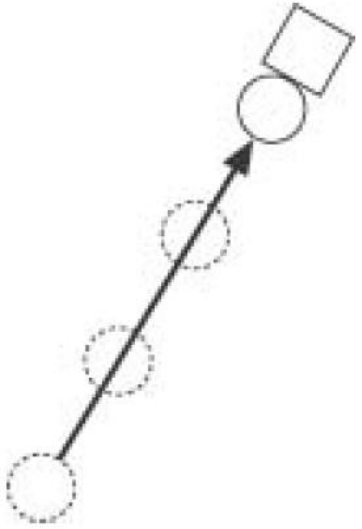
```
if (predatorX > preyX) {
   predatorX--;
} else if (predatorX == preyX) {
   // do nothing
} else {
   predatorX++;
}

if (predatorY > preyY) {
   predatorY--;
} else if (predatorY == preyY) {
   // do nothing
} else {
   predatorY++;
}
```



2. Chasing and Evading

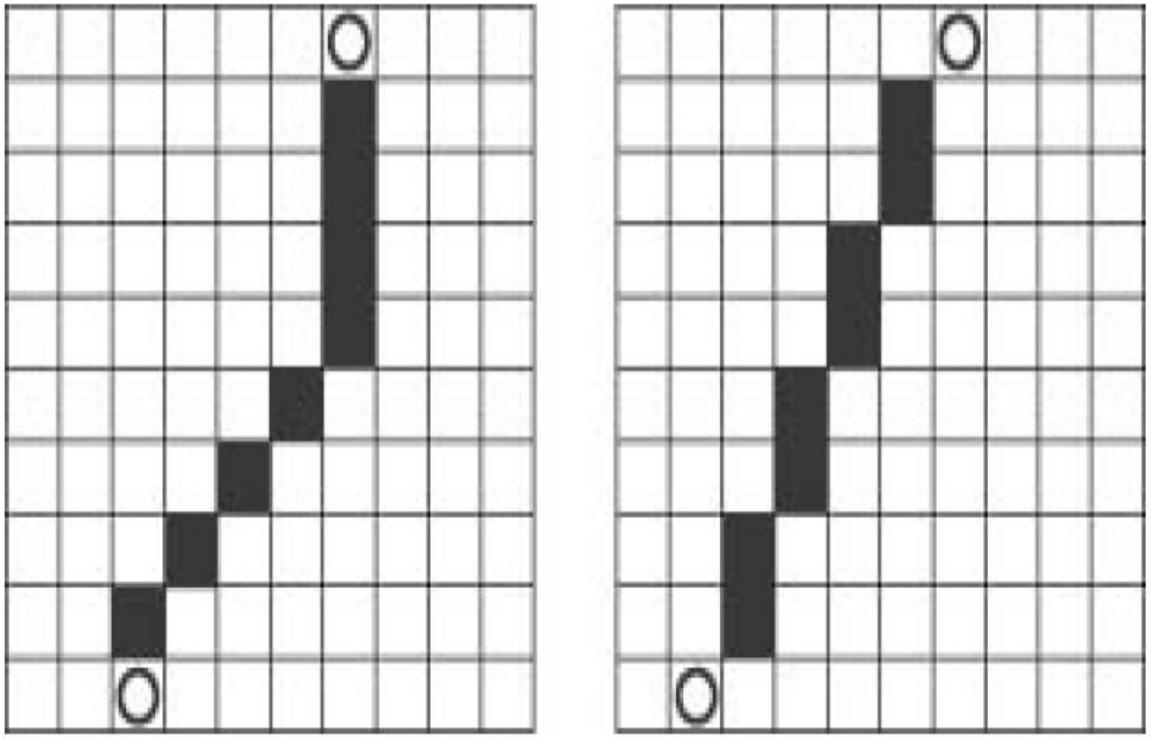
Sin obstáculos: Line-of-Sight Chasing

La esencia de este enfoque es que el cazador sigua una línea recta hacia la presa. El cazador siempre se mueve directamente a la posición de la presa.



2. Chasing and Evading

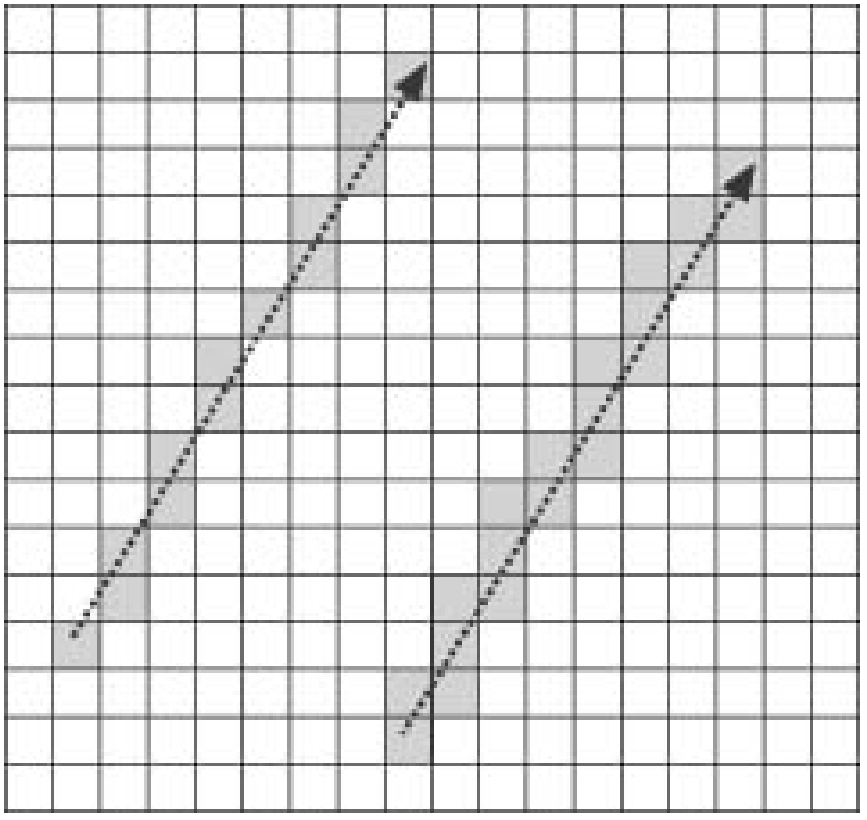
Simple VS Line-of-Sight Chasing



2. Chasing and Evading

Algoritmo de Bresenham

Bresenham vs anteriores:



Inteligencia Artificial en Videojuegos

TEMARIO

1. Introducción
2. Chasing and evading
3. **Patrones de movimiento**
4. Focking
5. Búsqueda de caminos
6. Máquinas de estado finito
7. Fuzzy logic
8. IA basada en reglas

3. Patrones de Movimiento

Es una forma simple de dar la ilusión de comportamiento inteligente.

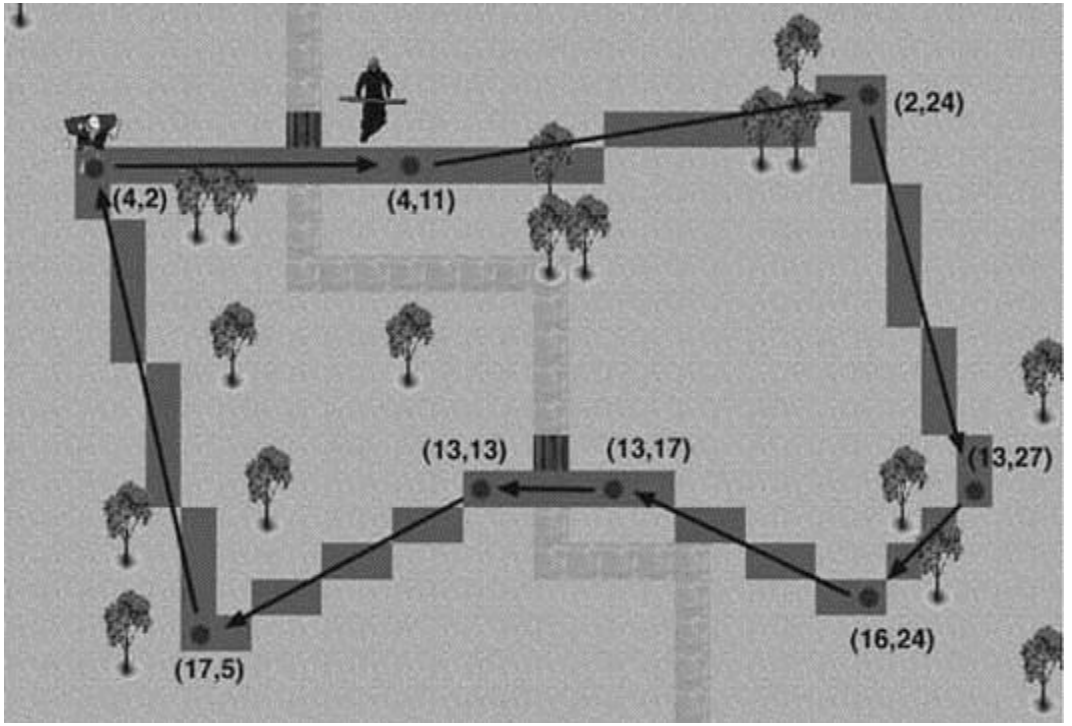
Los NPC se mueven según un algoritmo de movimiento predefinido que da la impresión de que realizan maniobras pensadas.

3. Patrones de Movimiento

- Comúnmente se codifica el patrón deseado en un array o matriz de instrucciones simples de movimiento.
- Las instrucciones simples pueden ser del tipo mover, girar, saltar, disparar, etc.
- Para procesar el patrón se debe mantener un índice, las instrucciones a procesar son las que corresponden a la posición del array indicada por el índice. También se debe tener una condición para avanzar en el índice.
- Es común tener distintos patrones para el mismo personaje. Luego la elección del patrón a utilizar puede ser aleatoria o en base a la lógica del juego.

3. Patrones de Movimiento

Los movimientos pueden representarse también por segmentos de líneas, donde el fin de uno es el comienzo del siguiente.



3. Patrones de Movimiento

- En juegos donde el escenario se represente como un tile map se puede crear un tile map adicional con los caminos posibles. Por ejemplo una matriz con 0s y 1s donde un 1 representa un lugar hacia donde el personaje puede moverse.
- Se pueden crear caminos alternativos entre 2 puntos, y realizar la selección de la próxima celda a visitar de forma aleatoria para que los personajes no repitan siempre los mismos caminos.

3. Patrones de Movimiento

- En juegos con simulación física obligar a los NPC a seguir un determinado movimiento predefinido puede implicar pérdida de realismo.
- La mejor opción en esos casos es aplicar fuerzas al objeto para moverlo hacia donde queremos que valla dejando que la simulación física determine la siguiente posición.
- El array de movimientos tendrá información de que fuerzas aplicar al NPC y cuanto tiempo hacerlo. Las condiciones de cuanto tiempo hacerlo podrían ser por ejemplo hasta llegar a cierto punto o hasta recorrer cierta distancia.

Inteligencia Artificial en Videojuegos

TEMARIO

1. Introducción
2. Chasing and evading
3. Patrones de movimiento
4. **Focking**
5. Búsqueda de caminos
6. Máquinas de estado finito
7. Fuzzy logic
8. IA basada en reglas

4. Flocking

Se refiere a cierto tipo de movimiento y comportamiento de determinados animales al desplazarse o al cazar.

Es una especie de movimiento en manada, podría pasar que en un videojuego se quiera implementar el movimiento de un conjunto de agentes de forma coordinada y real.

4. Flocking

Existe un método desarrollado por Craig Reynold's que implementa un flocking sin lider que esta basado en las siguientes tres reglas:

- Cohesión: Cada unidad se mueve hacia la posición media de sus vecinos.
- Alineación: Cada unidad se mueve para alinearse con la cabeza de sus vecinos.
- Separación: Cada unidad evita golpear sus vecinos.

Parámetros:

- Visibilidad: El radio de movimiento a considerar.
- Campo de vista:
  - Sólo cierto ángulo a considerar.
  - Estrecho campo de visión(hormigas).

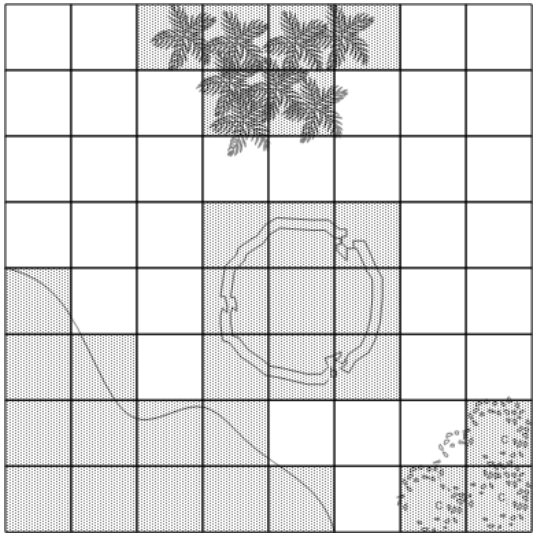
Inteligencia Artificial en Videojuegos

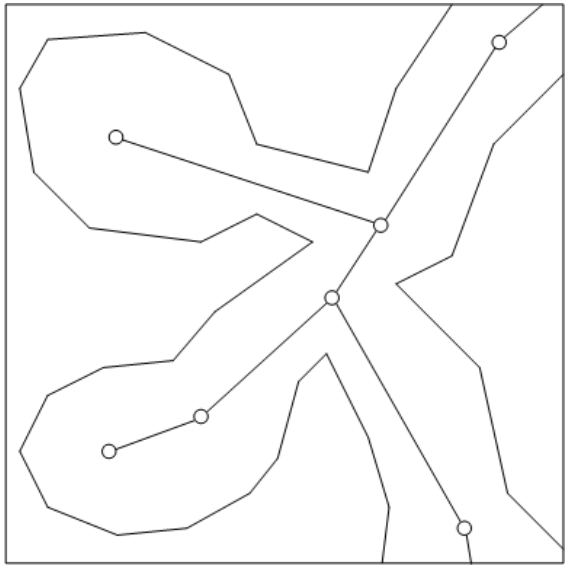
TEMARIO

1. Introducción
2. Chasing and evading
3. Patrones de movimiento
4. Focking
5. **Búsqueda de caminos**
6. Máquinas de estado finito
7. Fuzzy logic
8. IA basada en reglas

5. Búsqueda de caminos

La búsqueda de caminos trata acerca de la forma de encontrar y seguir un trayecto en el universo del juego por parte de un agente.
Para lograr esto es necesario contar con una representación del entorno donde el agente se podrá desplazar, puede ser una grilla, un conjunto de puntos, una superficie.





5. Búsqueda de caminos

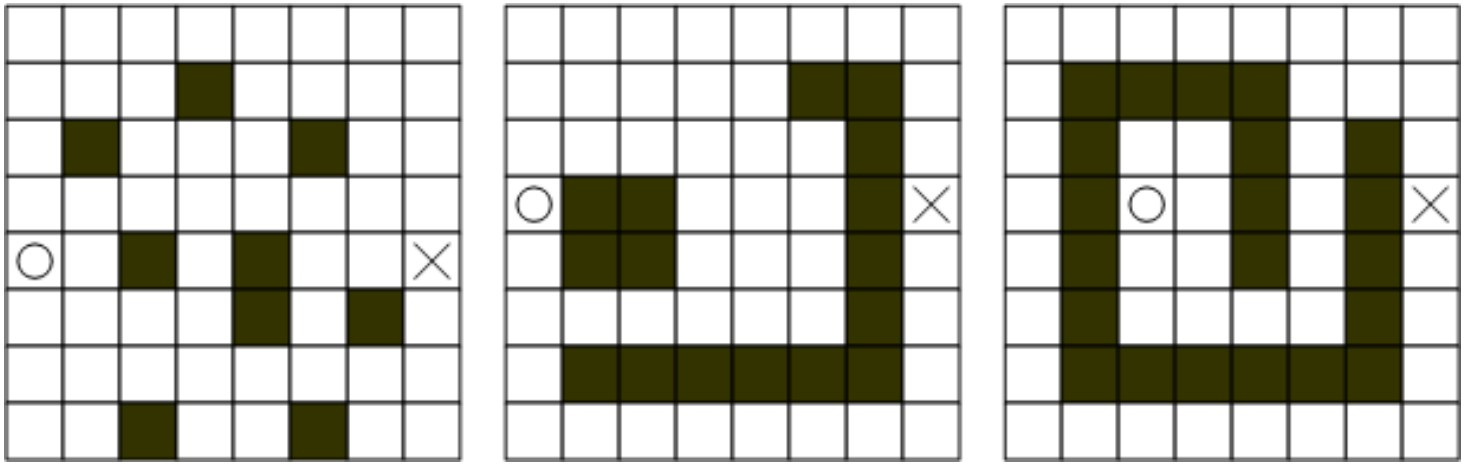
Un camino es una lista de puntos, celdas, nodos que el agente debe pasar. Un algoritmo de búsqueda de caminos se encarga de encontrar este camino, con criterios tales como:

Naturalidad de movimiento
Camino más corto
Tiempo de procesamiento

5. Búsqueda de caminos

Algoritmo Simple

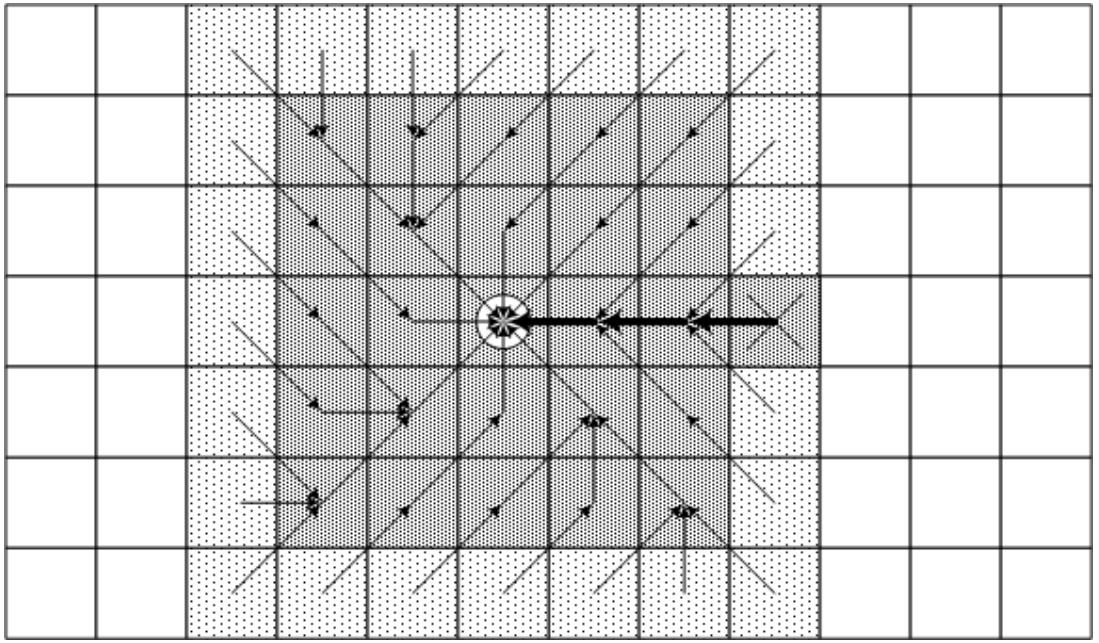
1. El agente se mueve una posición hasta alcanzar el objetivo.
2. Si lo encuentra termina
3. Si encuentra un obstáculo, lo rodea(sentido horario o antihorario) para esquivarlo
4. Si no encuentra el objetivo vuelve al paso 1.



5. Búsqueda de caminos

Algoritmo Breadth-First

Se parte de una posición inicial, y se buscan todos los caminos posibles, hasta encontrar una solución. La representación del terreno puede ser mediante grilla o nodos.



5. Búsqueda de caminos

Algoritmo Breadth-First

1. Se elije el nodo inicial y se hace un push en la lista de nodos con este elemento.
2. Mientras la lista de nodos no sea vacía
    A. Se hace Pop de un nodo(currentNode)
    B. Si currentNode es el objetivo terminar
    C. Crear nuevos nodos(successors nodes) para cada una de las celdas adyacentes al currentNode a y hacer push para cada uno de estos en la lista.
    D. Push currentNode en la lista de recorrida(close list)

5. Búsqueda de caminos

Algoritmo Dijkstra o camino mínimo

PASO 1:

$$D_s = 0$$

$$D_j = d_{s,j} \forall j <> s$$

$$P = \{s\}$$

$$T = \{resto\_nodos\}$$

PASO 3:

$$\forall j \notin P, D_j = min\{D_j, d_{i,j} + D_i\}$$

Condiciones iniciales

- *S={1}*
- *V-S={2,3,4,5}*
- *D=[0,2,1,∞,3]*
- 1 2 3 4 5

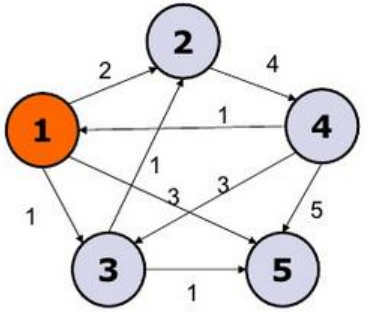
PASO 2:

Encontramos $i \notin P | D_i = \min_{j \notin P}\{D_j\}$

Puede existir un empate, en tal caso se elige uno arbitrariamente o de acuerdo a un criterio marcado

$$P = P \cup \{i\}$$

Si P contiene a todos los nodos se para, si no continuamos con el paso 3.



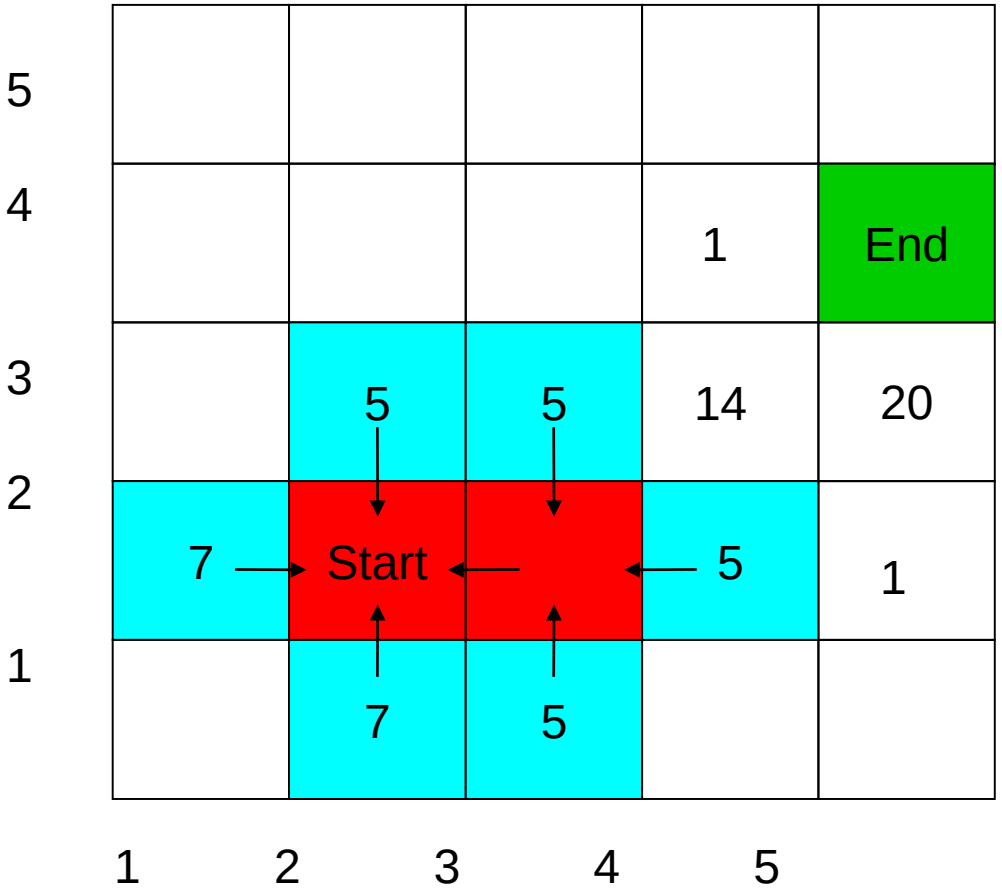
5. Búsqueda de caminos

Algoritmo A*

```
Se agrega el nodo inicial a la lista abierta

while (not_empty(open_list)) {
    current_node := node from open_list with lowest cost
    if (current_node==goal_node) {
B ——→   path complete
    } else {
        move current_node to closed_list
        for each node adjacent to current_node {
            if ((node is not in open_list) && (node is not in closed_list)) {
A ——→           move node to open_list
                assign cost to node
            }
        }
    }
}
```

Cuando un nodo es agregado a la open list linea A, mantiene una referencia a su padre. Su padre es el current_node.
Al final cuando se encuentra el camino(line B), siguiendo el puntero hacia el padre es como se encuentra la traza del camino óptimo.

5. Búsqueda de caminos



Iteración 3:

Open List:

Nodo: (1,2)
Costo: 7
Distancia del orígen: 1

Nodo: (2,1)
Costo: 7
Distancia del orígen: 1

Nodo: (2,3)
Costo: 5
Distancia del orígen: 1

Nodo: (3,3)
Costo: 5
Distancia del orígen: 2

Nodo: (4,2)
Costo: 5
Distancia del orígen: 2

Nodo: (3,1)
Costo: 7
Distancia del orígen: 2

Closed List:

Nodo: (2,2)

Nodo: (3,2)

Inteligencia Artificial en Videojuegos

TEMARIO

1. Introducción
2. Chasing and evading
3. Patrones de movimiento
4. Focking
5. Búsqueda de caminos
6. **Máquinas de estado finito**
7. Fuzzy logic
8. IA basada en reglas

6. Máquinas de estado finitas

Máquinas de estados deterministas

Cada NPC puede estar en un conjunto finito de estados posibles (persiguiendo, huyendo, patrullando, etc.) a lo largo de su vida, y en determinado momento solo se encuentra en uno de ellos. Con esta idea se puede modelar el comportamiento de los NPC utilizando FSM.

Como ventaja su implementación es sencilla, ya que dividen cada comportamiento en distintos fragmentos de código, en general se almacena la información en estructuras de datos tipo tablas, o en POO se crean clases que representan a los estados, y sus métodos están asociados alas transiciones.

Para modelar el comportamiento mediante una FSM, se deben definir: Estados, Transiciones, acciones.

6. Máquinas de estado finitas

Máquinas de estados deterministas

Estados: definen el comportamiento actualdel agente y pueden almacenar información del pasado.

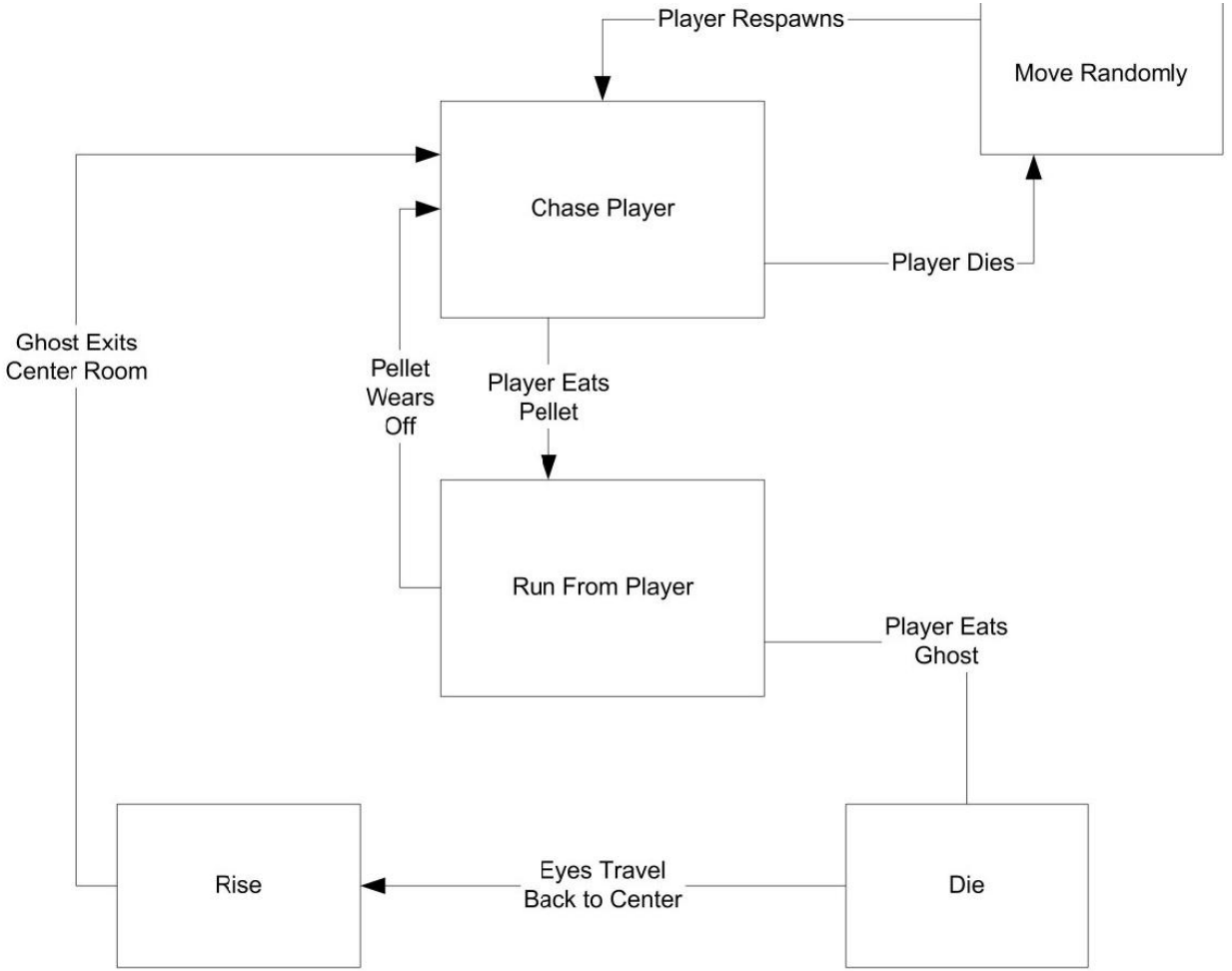
Transiciones: describen el cambio de estado, y establecen las condiciones para llegar al nuevo estado.

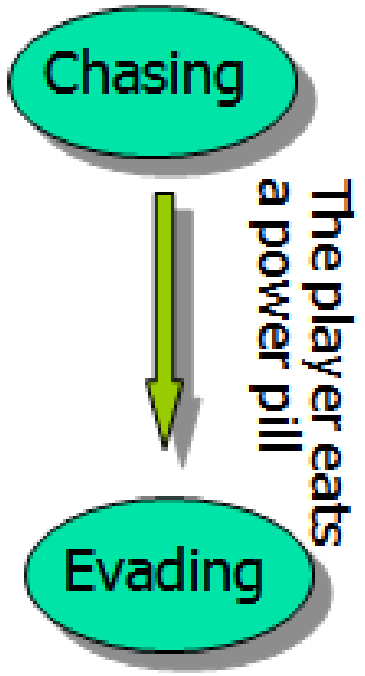
Acciones:

- De entrada, que se ejecutan cuando se entra en el nuevo estado.
- De salida, ejecutadas cuando se sale de ese estado.
- Inputs actions, ejecutadas según los cambios en el estado del juego, vinculadas al estado actual.
- De transición, que se ejecutan cuando se produce determinada transición.

6. Máquinas de estado finitas

Máquinas de estados deterministas





6. Máquinas de estado finitas

Máquinas de estados deterministas

Formas de implementación:

- Hardcoded
- Scripting
- Enfoque Hibrido

6. Máquinas de estado finitas

Máquinas de estados deterministas

Formas de implementación

Hardcoded

```
void Step(int *state){
    switch(state) {
         case 0:  // pensar
            Wander();
            if( SeeEnemy() )    { *state = 1; }
            break;

        case 1:  // atacar
            Attack();
            if( LowOnHealth() ) { *state = 2; }
            if( NoEnemy() )     { *state = 0; }
            break;
        case 2:  // rajar
            Flee();
            if( NoEnemy() )     { *state = 0; }
            break;
    }
}
```

Problemas con este enfoque:

- Código a fuego.
- Las transiciones son ejecutadas utilizando polling ya que en cada iteración se pregunta por todos los eventos.
- No puede ser modificado por el usuarios.
- No puede determinar el primer evento ocurrido.

Alternativa: Implementar los estados como objetos, donde las transiciones sean eventos

6. Máquinas de estado finitas

Máquinas de estados deterministas

Formas de implementación

Scripting

```
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 )
}
```

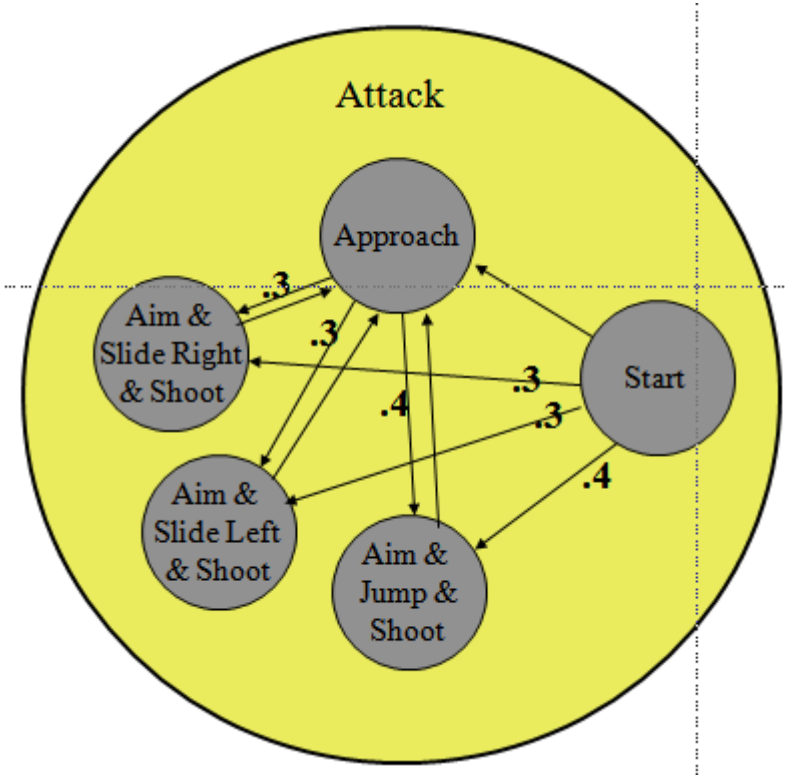
Ventajas
Se utilizan eventos
Existen conceptos tales como
OnEnter and OnExit

Desventajas
No es trivial la implementación

6. Máquinas de estado finitas

Máquinas de estados no deterministas

Existen múltiples transiciones para el mismo evento.
Cada transición se encuentra etiquetada con un valor que indica la probabilidad de que suceda.
Al ocurrir un evento aleatoriamente en tiempo de ejecución se selecciona la transición. Esto hace que sea más difícil el debug.
En el modelo de Markov, el estado actual solo depende del anterior.
Según el modelo podría estar en más de un estado a la vez.



Inteligencia Artificial en Videojuegos

TEMARIO

1. Introducción
2. Chasing and evading
3. Patrones de movimiento
4. Focking
5. Búsqueda de caminos
6. Máquinas de estado finito
7. **Fuzzy logic**
8. IA basada en reglas

7. Fuzzy Logic

- La lógica difusa se basa en presentar los problemas en una forma similar a como lo resuelven los humanos. Esto es de forma imprecisa, sin usar medidas exactas, sin tomar en cuenta todos los factores o generalizando.
- Los problemas se plantean y resuelven usando expresiones similares al lenguaje natural.
  - Cerca, lejos, muy lejos. Alto, bajo, muy alto, muy bajo.
- Se basa en que no todo es totalmente cierto o falso. Hay una graduación de verdad en la cual el 0 es el falso absoluto y el 1 la verdad absoluta, pero una sentencia puede tener cualquier valor de verdad entre 0 y 1.

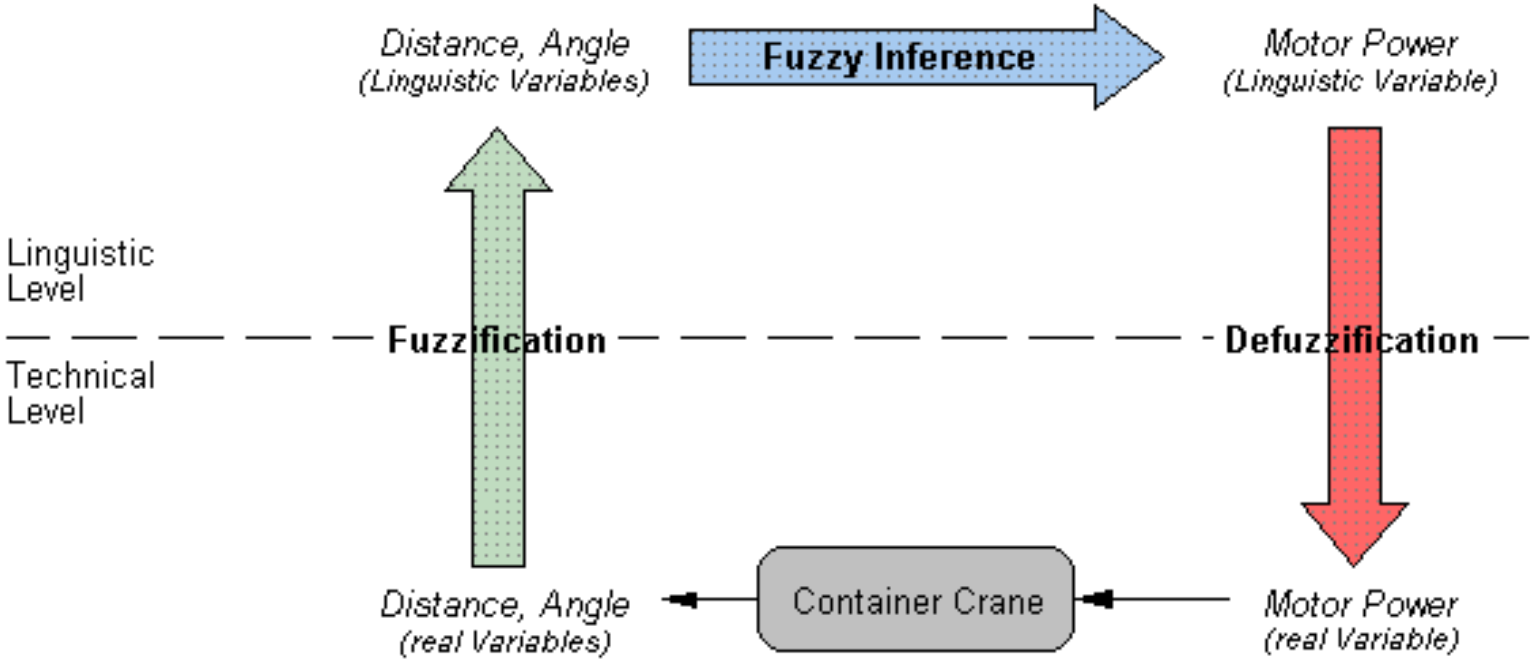
7. Fuzzy Logic

- Permite modelar mejor cambios suaves de decisiones dado cambios en los factores que se toman en cuenta. En la lógica booleana generalmente los cambios se dan de forma abrupta, y para que no sea así se deben dividir las entradas del problema en intervalos pequeños.
- Permite modelar también un razonamiento no tan perfecto y menos predecible.
- Puede ser usada en control, evaluación de amenazas, clasificación.

7. Fuzzy Logic

Proceso de lógica difusa:

- Fuzzification
- Fuzzy inference
- Desfuzzification



7. Fuzzy Logic

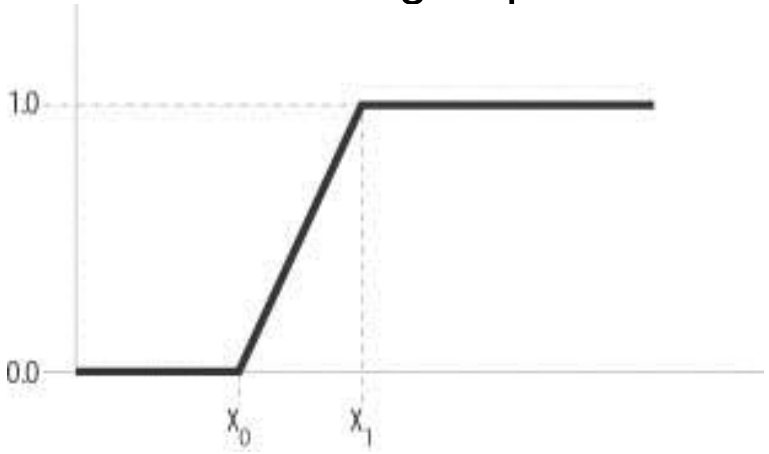
Fuzzification:

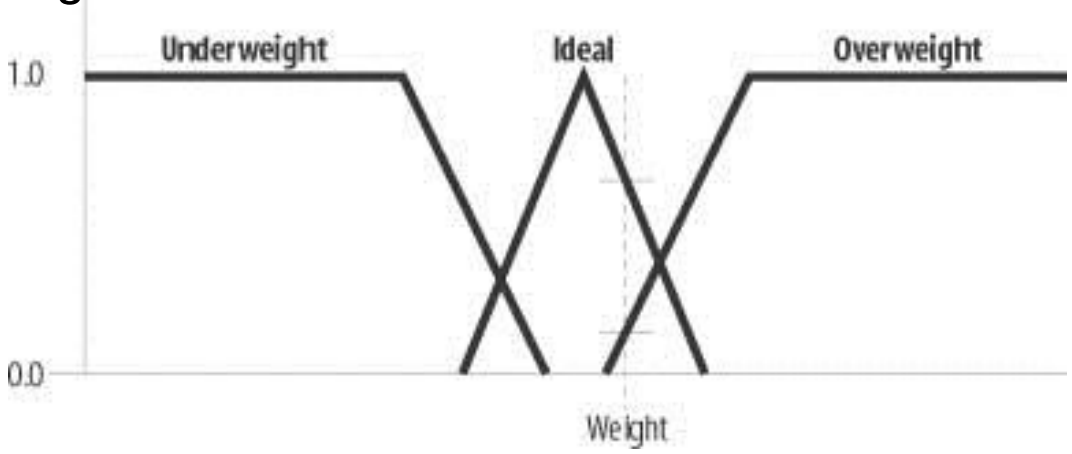
- Se toman las medidas reales y se pasa a valores difusos.

  Peso en kg → {sobrepeso, peso normal, bajo peso}

- Para todo valor difuso debe existir una función de pertenencia. Esta indica el grado en que el valor real tomado como entrada se mapea o no al valor difuso.

  70 kg -> peso ideal con grado 0.75





7. Fuzzy Logic

Fuzzy inference:

- Se trata de aplicar reglas de lógica difusa para determinar el grado de verdad de cada regla.

  if sobrepeso and no activo then realizar ejercicio

  if sobrepeso and activo then cambiar la dieta

- Axiomas:

  Verdad (a OR b) = Max (Verdad(a), Verdad(b))

  Verdad (a AND b) = Min (Verdad(a), Verdad(b))

  Verdad (NOT a) = 1 - Verdad(a)

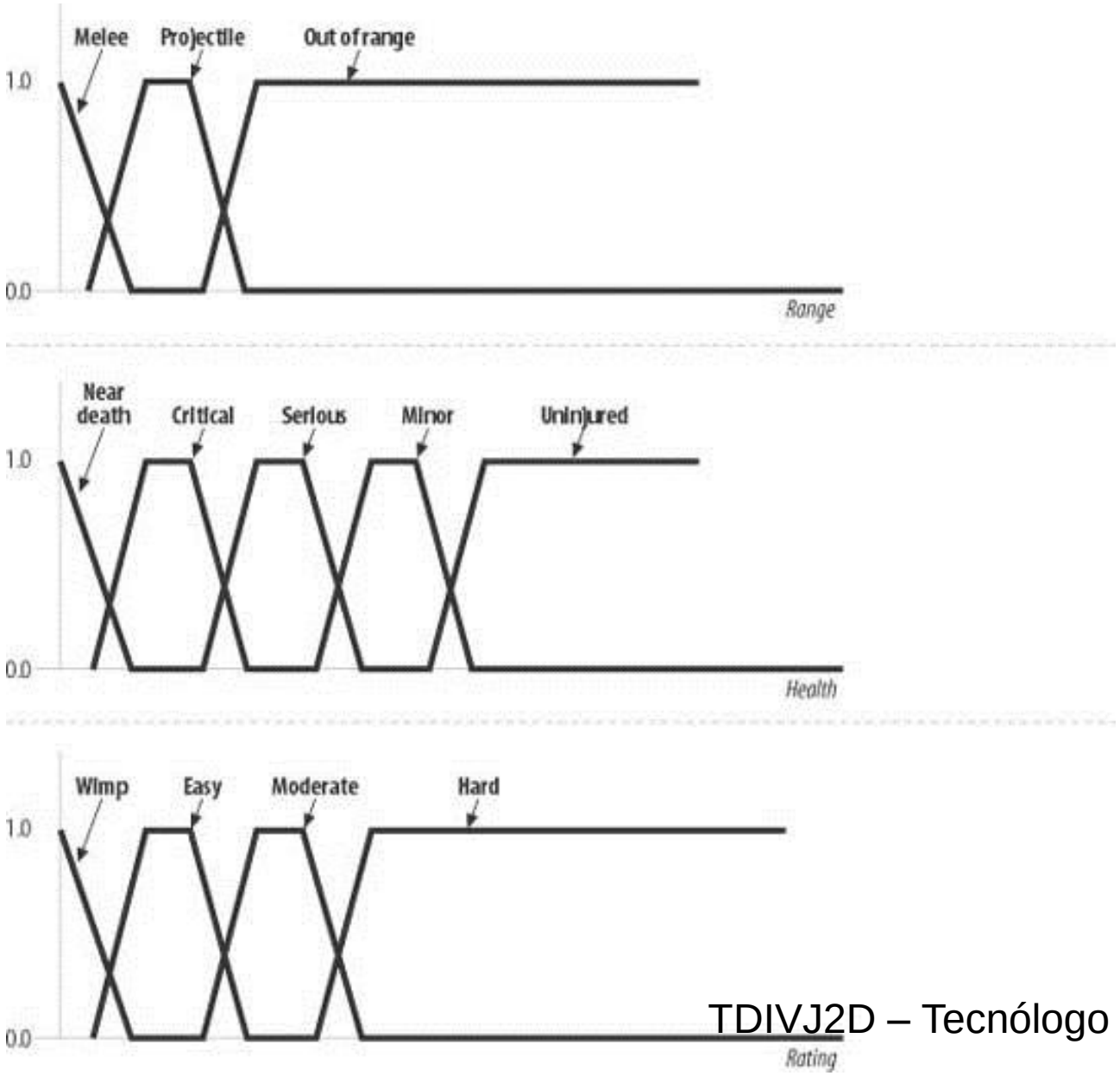
7. Fuzzy Logic

Fuzzy inference:

- En lógica booleana las reglas se evalúan en forma serial hasta que una da true.
- En lógica difusa todas las reglas deben ser evaluadas.
- El valor obtenido es la fuerza de cada regla.

7. Fuzzy Logic

Ejemplo:



7. Fuzzy Logic

Ejemplo:

- Reglas:

  if (in melee range AND uninjured) AND NOT hard then attack

  if (NOT in melee range) AND uninjured then do nothing

  if (NOT out of range AND NOT uninjured) AND (NOT wimp) then flee

- Un posible resultado:

  attack=2.0, do nothing=0.4, flee=0.7

7. Fuzzy Logic

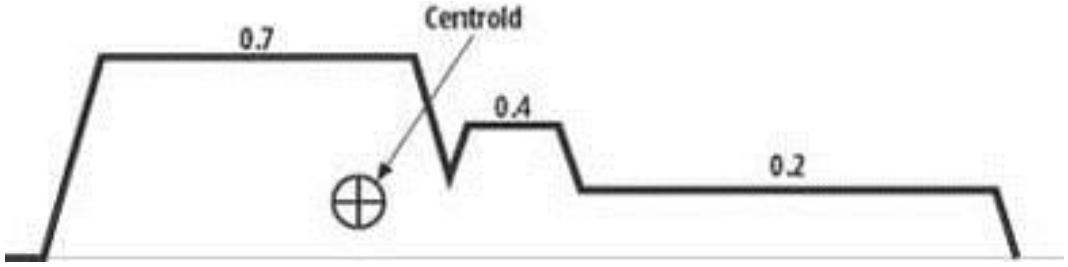
Desfuzzification:

- Se toman los valores de los datos difusos y se traducen a valores reales.
- Se hace con una función llamada de des-pertenencia (output membership).
- En estas se toman en cuenta el grado de pertenencia a cada valor obteniendo la función de pertenencia compuesta la cual es la unión de la función de pertenencia a cada valor difuso truncada por la fuerza calculada de cada regla.

7. Fuzzy Logic

Desfuzzification:

- El método más común para determinar el valor a devolver es hallar el centroide de la figura resultante y retornar su valor en x.



- Otro método es asignar valores a cada valor difuso y calcular el promedio ponderado de los valores obtenidos para cada regla:
  - flee = -10, do-nothing = 1, attack = 10.
  - [10*0.2 + 1*0.4 - 10*0.7] / (0.2+0.4+0.7) = -2.98

Inteligencia Artificial en Videojuegos

TEMARIO

1. Introducción
2. Chasing and evading
3. Patrones de movimiento
4. Focking
5. Busqueda de caminos
6. Máquinas de estado finito
7. Fuzzy logic
8. **IA basada en reglas**

8. IA basada en reglas

- Es probablemente el sistema de IA más utilizado. Consisten en un conjunto de reglas de estilo *if then* que se utilizan para hacer inferencias o tomar decisiones de acción.
- Las máquinas de estados finitos y la lógica difusa caen dentro de esta categoría.
- Una de sus ventajas es que imita la manera como la gente tiende a pensar.
- Otra ventaja es que es bastante fácil de programar porque el conocimiento codificado en las reglas es modular y las reglas pueden ser codificadas en cualquier orden.

8. IA basada en reglas

Componentes principales:

- **Memoria de trabajo:** almacena los datos conocidos y las afirmaciones hechas por las reglas.
- **Memoria de reglas:** contiene normas del estilo *If then* que operan sobre los datos almacenados en la memoria de trabajo. Como las reglas se activan, o despedido en basado en normas
- **Afirmaciones:** nueva información introducida por las reglas. Se guardan en la memoria de trabajo

Las reglas pueden desencadenar acciones, cambios de estado o modificar el contenido de la memoria de trabajo.

8. IA basada en reglas

Inferencias

Hay dos mecanismos principales para realizar inferencias a partir de las reglas:

- Encadenamiento hacia adelante.
- Encadenamiento hacia atrás.

8. IA basada en reglas

Encadenamiento hacia adelante:

- La primera fase consiste comprobar los *if* de cada regla para ver si se cumplen con los hechos y afirmaciones de la memoria de trabajo.
- Si más de una regla se activa hay que averiguar que regla disparar. Esta es la fase de resolución de conflictos.
- La decisión de que regla disparar se puede hacer de muchas maneras: la primera regla coincidente, una al azar, ponderarlas, etc.
- Por último se ejecuta la parte *then* de la regla seleccionada.

8. IA basada en reglas

Encadenamiento hacia atrás:

- Se comienza por algún resultado o meta, y se averigua que reglas deben ser disparadas para llegar a esa meta.

8. IA basada en reglas

Predicción:

- Los sistemas de inteligencia artificial basados en reglas pueden enriquecerse mediante la inclusión de predicción como una forma de simular aprendizaje.
- En la memoria de trabajo se debe guardar un historial de las últimas acciones o movimientos del jugador.
- En base a esa información se puede determinar cual es la acción futura más probable del jugador y elegir la del NPC basándose en eso.

Referencias

- **AI for Game Developers - David M. Bourg, Glenn Seeman - O'Reilly 2004.**