

Inteligencia Artificial

Inferencia en lógica proposicional



[Transparencias adaptadas de Dan Klein and Pieter Abbeel: CS188 Intro to AI, UC Berkeley (ai.berkeley.edu)]

Índice

- **Tema anterior**

- Conceptos básicos de conocimiento, lógica, razonamiento
- Lógica proposicional: sintaxis y semántica
- Ejemplos en el mundo del Wumpus
- La base de conocimiento de un Pacman parcialmente observable

- **Este tema**

- Inferencia por comprobación de modelos
- Agentes lógicos: Pacman
- Inferencia por demostración de teoremas (brevemente)

Repaso: Ideas básicas

- **Base de conocimiento (KB)** = conjunto de *sentencias* (que representan afirmaciones sobre el mundo) en un lenguaje formal (lógica)
- Los agentes lógicos se basan en demostrar implicación entre sentencias de la base de conocimiento.
- **Implicación:** $\alpha \models \beta$ (“ α implica β ” o “ β se sigue de α ”) si y sólo si en todos los mundos donde α es verdad, β también es verdad
 - Es decir, los mundos α son un subconjunto de los mundos β :

$$M(\alpha) \subseteq M(\beta)$$

Repaso: Semántica de la lógica proposicional

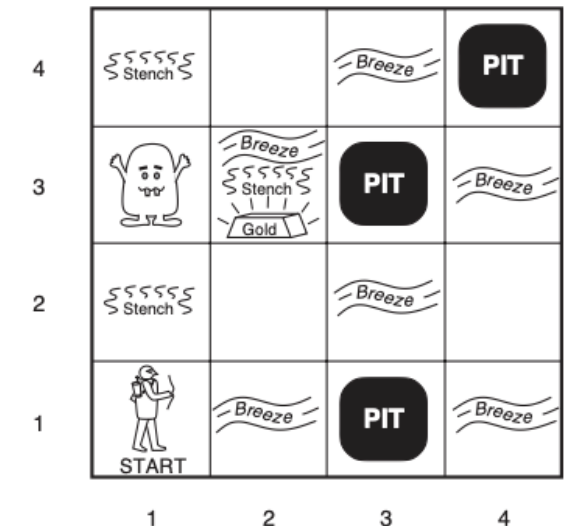
- Sea m un modelo que asigna verdadero o falso a $\{X_1, X_2, \dots, X_n\}$ (definición de modelo)
- Si α es un símbolo, entonces su valor de verdad se da en m
- $\neg\alpha$ es verdad en m sii α es falso en m
- $\alpha \wedge \beta$ es verdad en m sii α es verdad en m y β es verdad en m
- $\alpha \vee \beta$ es verdad en m sii α es verdad en m o β es verdad en m
- $\alpha \Rightarrow \beta$ es verdad en m sii α es falso en m o β es verdad en m
- $\alpha \Leftrightarrow \beta$ es verdad en m sii $\alpha \Rightarrow \beta$ es verdad en m y $\beta \Rightarrow \alpha$ es verdad en m

Repaso: Inferencia

- Método 1: *comprobación de modelos*
 - Para cada mundo posible, si α es verdad, asegura que β también lo sea
- Método 2: *demostración de teoremas*
 - Búsqueda de una secuencia de pasos de prueba (aplicación de las reglas de inferencia) que lleven de α a β
- Algoritmo **sólido**: deriva solo sentencias implicadas (no se inventa cosas)
- Algoritmo **completo**: puede derivar cualquier sentencia que esté implicada

Repaso: Comprobación de modelos

- $P_{x,y}$ es verdadero si hay un pozo en $[x, y]$.
- $W_{x,y}$ es verdadero si hay un Wumpus en $[x, y]$, vivo o muerto.
- $B_{x,y}$ es verdadero si el agente percibe una brisa en $[x, y]$.
- $S_{x,y}$ es verdadera si el agente percibe un hedor en $[x, y]$.



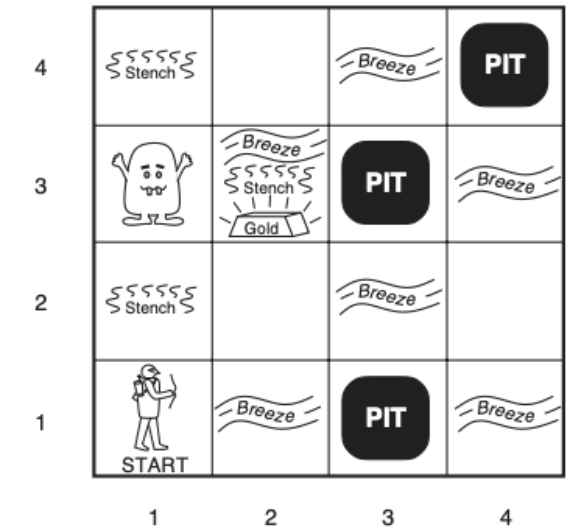
Sentencias para derivar $\neg P_{1,2}$

- No puede haber pozo en la casilla de salida
 - $R_1: \neg P_{1,1}$
- Por cada casilla, establecemos la condición de que haya brisa
 - $R_2: B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
 - $R_3: B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
- Las percepciones que llevamos hasta el momento
 - $R_4: \neg B_{1,1}$
 - $R_5: B_{2,1}$

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

Repaso: Comprobación de modelos

- El objetivo es decidir $KB \models \alpha$ para algún α
- Por ejemplo, pongamos que α es $\neg P_{1,2}$
- Veamos los símbolos relevantes: $B_{1,1}, B_{2,1}, P_{1,1}, P_{1,2}, P_{2,1}, P_{2,2}, P_{3,1}$
- KB es verdadero si $R_1 \dots R_5$ son verdaderas
- **Si en los modelos en los que la KB es verdadera, α también lo es, entonces $KB \models \alpha$**



$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	BC
falso	falso	falso	falso	falso	falso	falso	verdadero	verdadero	verdadero	verdadero	falso	falso
falso	falso	falso	falso	falso	falso	verdadero	verdadero	verdadero	falso	verdadero	falso	falso
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
falso	verdadero	falso	falso	falso	falso	falso	verdadero	verdadero	falso	verdadero	verdadero	falso
falso	verdadero	falso	falso	falso	falso	verdadero	verdadero	verdadero	verdadero	verdadero	verdadero	verdadero
falso	verdadero	falso	falso	falso	verdadero	falso	verdadero	verdadero	verdadero	verdadero	verdadero	<u>verdadero</u>
falso	verdadero	falso	falso	falso	verdadero	verdadero	verdadero	verdadero	verdadero	verdadero	verdadero	<u>verdadero</u>
falso	verdadero	falso	falso	falso	falso	verdadero	verdadero	verdadero	verdadero	verdadero	verdadero	<u>verdadero</u>
falso	verdadero	falso	falso	falso	falso	falso	verdadero	verdadero	verdadero	verdadero	verdadero	verdadero
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
verdadero	verdadero	verdadero	verdadero	verdadero	verdadero	verdadero	falso	verdadero	verdadero	falso	verdadero	falso

Algoritmo basado en tabla de verdad (TT)

- Para decidir implicación proposicional
- Sólido y completo

function TT-ENTAILS?(KB, α) **returns** *true or false*
inputs: KB , the knowledge base, a sentence in propositional logic
 α , the query, a sentence in propositional logic

 $symbols \leftarrow$ a list of the proposition symbols in KB and α
return TT-CHECK-ALL($KB, \alpha, symbols, \{ \}$)

function TT-CHECK-ALL($KB, \alpha, symbols, model$) **returns** *true or false*
if EMPTY?($symbols$) **then**
 if PL-TRUE?($KB, model$) **then return** PL-TRUE?($\alpha, model$)
 else return *true* // when KB is false, always return *true*
else do
 $P \leftarrow$ FIRST($symbols$)
 $rest \leftarrow$ REST($symbols$)
 return (TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = true\}$)
 and
 TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = false\}$))

Satisfacibilidad e implicación

- Una sentencia es **válida** si es verdad en todos los mundos
- Una sentencia es **satisfacible** si es verdad en al menos un mundo
 - El problema de determinar la satisfacibilidad de sentencias en lógica proposicional se llama SAT
- Supongamos que tenemos un solucionador SAT eficiente; ¿cómo podemos usarlo para probar implicación?
 - $\alpha \models \beta$ (en nuestros agentes α es la KB y β lo que queremos probar)
 - sii $\alpha \Rightarrow \beta$ es verdadero en todos los mundos
 - sii $\neg(\alpha \Rightarrow \beta)$ es falso en todos los mundos
 - sii $\alpha \wedge \neg\beta$ es falso en todos los mundos; es decir, insatisfacible
- Por tanto, añadimos la conclusión negada ($\neg\beta$) a lo que sabemos (α), y probamos la (in)satisfacibilidad: reducción al absurdo
- Los solucionadores SAT eficientes operan en **forma normal conjuntiva**

Forma normal conjuntiva (CNF)

-
- Diagram illustrating the conversion of a sentence into Conjunctive Normal Form (CNF) using logical equivalences:
- Cada sentencia **se convierte en una conjunción (\wedge) de cláusulas**
 - Reemplazar el bicondicional por dos implicaciones
 - Cada cláusula es una **disyunción (\vee) de literales**
 - Cada literal es una **literal**
 - Reemplazar $\alpha \Rightarrow \beta$ por $\neg\alpha \vee \beta$
 - Convertir la expresión en una **conjunción de transformaciones estándar:**
 - Distribuir \vee sobre \wedge
- Examples of transformations:
- $A \Rightarrow (W \Leftarrow B)$
 - $A \Rightarrow ((W \Rightarrow B) \wedge (B \Rightarrow W))$
 - $\neg A \vee ((\neg W \vee B) \wedge (\neg B \vee W))$
 - $(\neg A \vee \neg W \vee B) \wedge (\neg A \vee \neg B \vee W)$

Forma normal conjuntiva (CNF)

- Cada sentencia puede expresarse como una **conjunción** (\wedge) de **clausulas**
- Cada clausula es una **disyunción** (\vee) de **literales**
- Cada literal es un símbolo o un símbolo negado
- Conversión a CNF por una secuencia de transformaciones estándar:
 - $A \Rightarrow (W \Leftrightarrow B)$
 - $A \Rightarrow ((W \Rightarrow B) \wedge (B \Rightarrow W))$
 - $\neg A \vee ((\neg W \vee B) \wedge (\neg B \vee W))$
 - $(\neg A \vee \neg W \vee B) \wedge (\neg A \vee \neg B \vee W)$

Equivalencias lógicas

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{Conmutatividad de } \wedge$$

$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{Conmutatividad de } \vee$$

$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{Asociatividad de } \wedge$$

$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{Asociatividad de } \vee$$

$$\neg(\neg\alpha) \equiv \alpha \quad \text{Eliminación de la doble negación}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{Contraposición}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \quad \text{Eliminación de la implicación}$$

$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{Eliminación de la bicondicional}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{Ley de Morgan}$$

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{Ley de Morgan}$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{Distribución de } \wedge \text{ respecto a } \vee$$

$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{Distribución de } \vee \text{ respecto a } \wedge$$

Solucionadores SAT eficientes

- DPLL (Davis-Putnam-Logemann-Loveland) es el núcleo de los solucionadores modernos
- Enumeración recursiva en profundidad de todos los modelos (árbol binario, búsqueda con BPP/DFS) con algunos extras:
 - **Terminación anticipada (early termination)**: parar si
 - Se satisfacen todas las cláusulas; p. ej., $(A \vee B) \wedge (A \vee \neg C)$ se satisface por $\{A=\text{verdadero}\}$ (da igual el valor de B y C)
 - Cualquier cláusula se vuelve falsa; p. ej., $(A \vee B) \wedge (A \vee \neg C)$ se vuelve falsa por $\{A=\text{falso}, B=\text{falso}\}$ (da igual el valor de C)
 - **Literales puros**: si todas las apariciones de un símbolo en cláusulas aún no satisfechas tienen el mismo signo, entonces le damos al símbolo ese valor
 - P. ej., A es puro y positivo en $(A \vee B) \wedge (A \vee \neg C) \wedge (C \vee \neg B)$ así que lo ponemos a verdadero
 - **Cláusulas unitarias**: si una cláusula se queda con un único literal, establecemos el valor del símbolo para satisfacer la cláusula
 - P. ej., si $A=\text{falso}$, $(A \vee B) \wedge (A \vee \neg C)$ se convierte en $(\text{falso} \vee B) \wedge (\text{falso} \vee \neg C)$; es decir, $(B) \wedge (\neg C)$
 - Satisfacer las cláusulas unitarias a menudo conduce a una mayor propagación, nuevas cláusulas unitarias, etc.

Algoritmo DPLL

- Recordatorio: comprobamos la satisfacibilidad de $\alpha \wedge \neg\beta$. Si no es satisfacible, podremos afirmar que $\alpha \models \beta$

function DPLL-SATISFIABLE?(*s*) **returns** *true* or *false*

inputs: *s*, a sentence in propositional logic

clauses \leftarrow the set of clauses in the CNF representation of *s*

symbols \leftarrow a list of the proposition symbols in *s*

return DPLL(*clauses*, *symbols*, { })

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

if every clause in *clauses* is true in *model* **then return** *true*

if some clause in *clauses* is false in *model* **then return** *false*

P, *value* \leftarrow FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)

if *P* is non-null **then return** DPLL(*clauses*, *symbols* – *P*, *model* \cup { *P*=*value* })

P, *value* \leftarrow FIND-UNIT-CLAUSE(*clauses*, *model*)

if *P* is non-null **then return** DPLL(*clauses*, *symbols* – *P*, *model* \cup { *P*=*value* })

P \leftarrow FIRST(*symbols*); *rest* \leftarrow REST(*symbols*)

return DPLL(*clauses*, *rest*, *model* \cup { *P*=*true* }) **or**

DPLL(*clauses*, *rest*, *model* \cup { *P*=*false* })

Solucionadores SAT en el mundo real

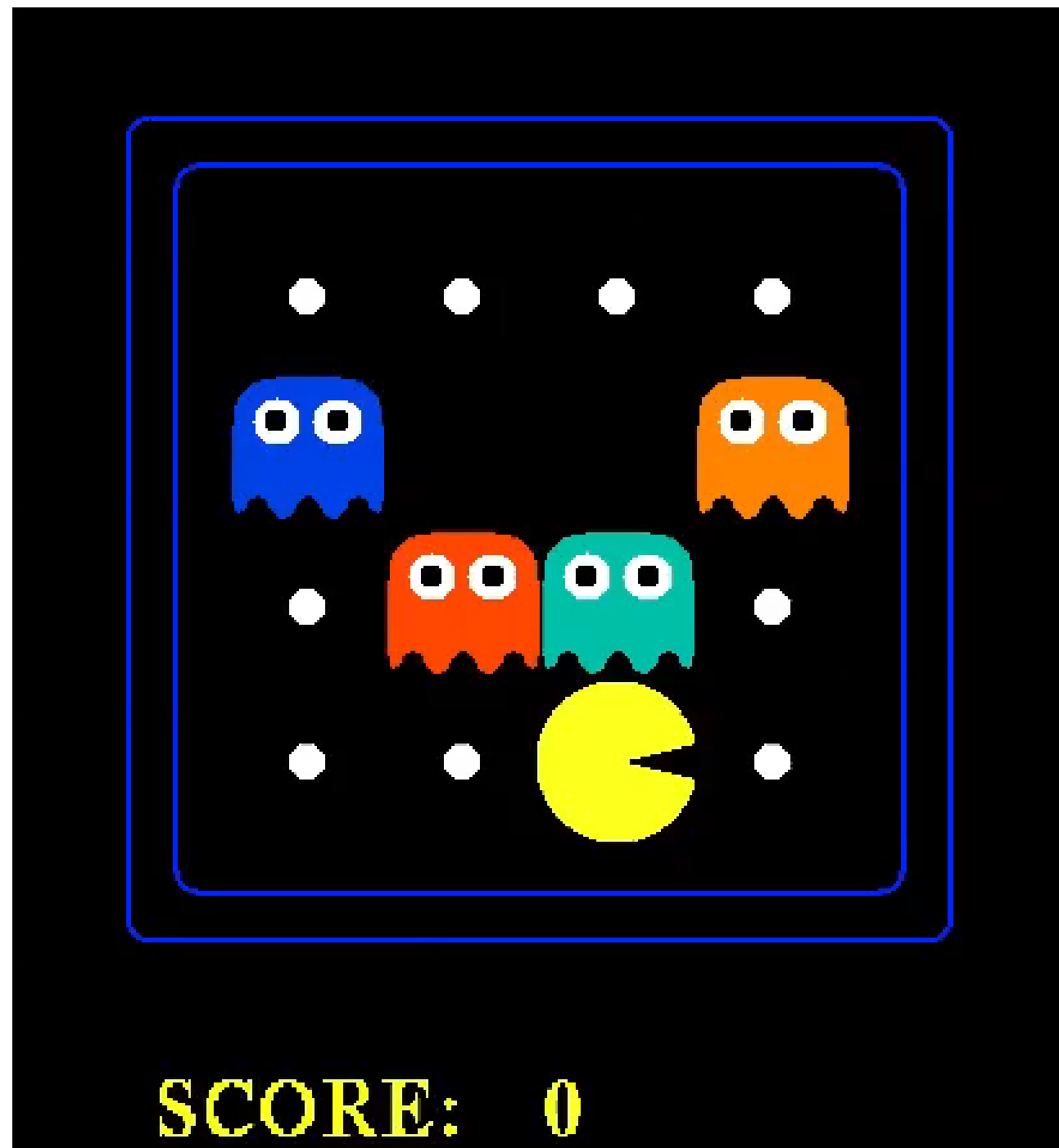
- Verificación de circuitos:
 - ¿Calcula este circuito VLSI la respuesta correcta?
- Verificación de software:
 - ¿Calcula este programa la respuesta correcta?
- Síntesis de software:
 - ¿Qué programa calcula la respuesta correcta?
- Verificación de protocolos:
 - ¿Se puede romper este protocolo de seguridad?
- Síntesis de protocolos:
 - ¿Qué protocolo es seguro para esta tarea?
- Muchos problemas combinatorios:
 - ¿Cuál es la solución?
- Planificación: **¿cómo puede Pacman comerse todos los puntos?**

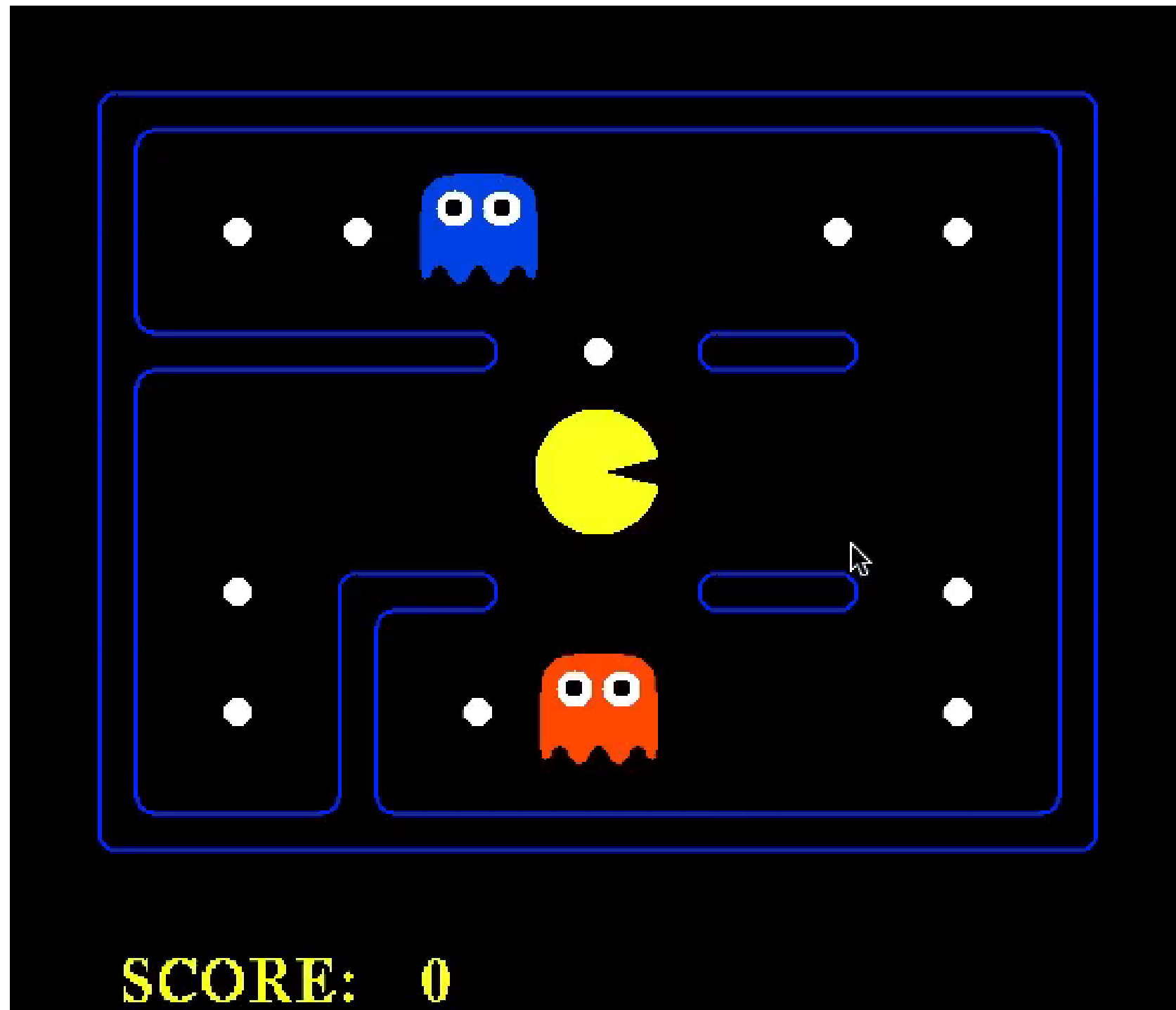
Planificación como satisfacibilidad

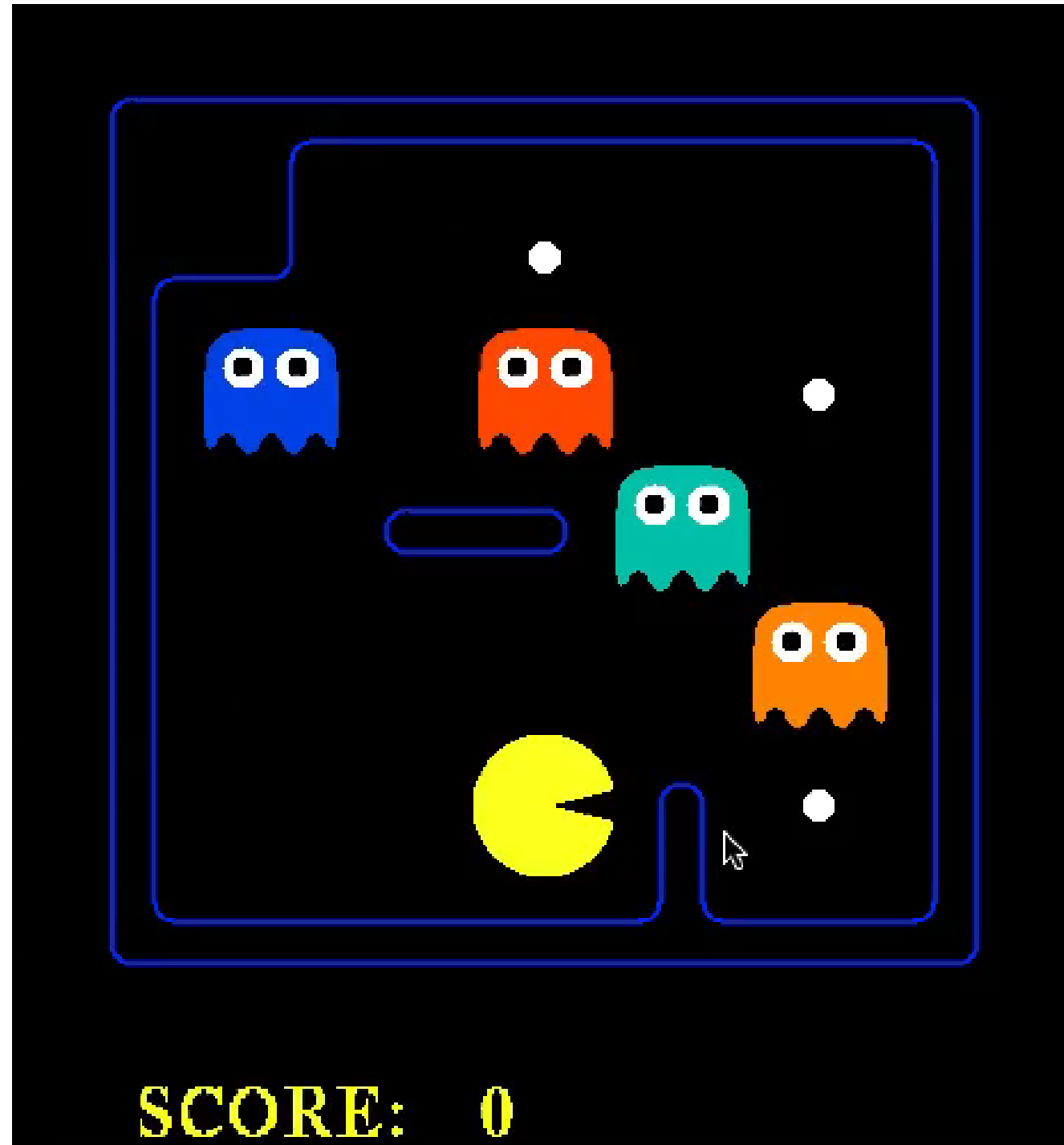
- Dado un solucionador SAT eficiente, ¿podemos utilizarlo para hacer planes?
- Sí, para el caso totalmente observable y determinista:
 - El problema de planificación puede resolverse si y solo si existe alguna asignación satisfactoria
 - Solución obtenida a partir de los valores de verdad de las variables de acción
- Para $T = 1$ a ∞ , (probamos si se puede resolver en 1 paso, si no, en 2, etc.)
 - Inicializar la KB con las físicas de Pacman (*PacFísicas*) para T pasos temporales
 - Afirmar que el objetivo es verdadero en el momento T
- Leer las variables de acción de la solución SAT

PacFísicas básicas para planificación

- **Mapa**: dónde están y dónde no están las paredes, **dónde está y dónde no está la comida**
- **Estado inicial**: lugar de inicio de Pacman (exactamente un lugar), **fantasmas**
- **Acciones**: Pacman hace exactamente una acción en cada paso
- **Modelo de transición**:
 - $\langle \text{en } x, y_t \rangle \Leftrightarrow [\text{en } x, y_{t-1} \text{ y se queda quieto}] \vee [\text{adyacente a } x, y_{t-1} \text{ y se mueve a } x, y]$
 - $\langle \text{comida } x, y_t \rangle \Leftrightarrow [\text{comida en } x, y_{t-1} \text{ y no se ha comido}]$
 - $\langle \text{fantasma}_B x, y_t \rangle \Leftrightarrow [\dots]$ (se mueven de acuerdo a una política fija)

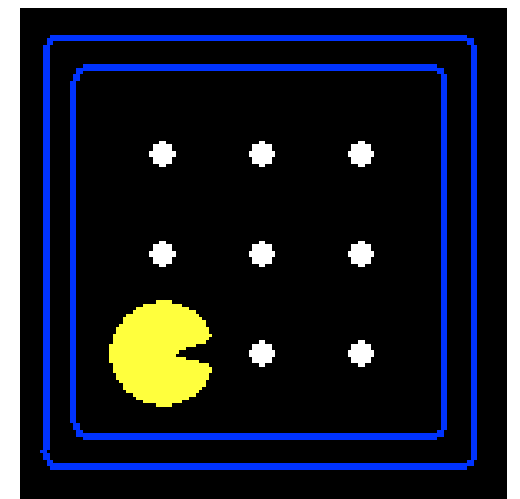






Recordatorio: Pacman parcialmente observable

- Percibe pared/no pared en cada dirección en cada momento
- Variables: Blocked_W_0 , Blocked_N_0 , ..., Blocked_W_1 , ...
- Modelo del sensor:
 - $\text{Blocked_W_0} \Leftrightarrow ((\text{At_1,1_0} \wedge \text{Wall_0,1}) \vee (\text{At_1,2_0} \wedge \text{Wall_0,2}) \vee (\text{At_1,3_0} \wedge \text{Wall_0,3}) \vee \dots)$
- Pregunta básica: ¿dónde estoy?




Estimación del estado

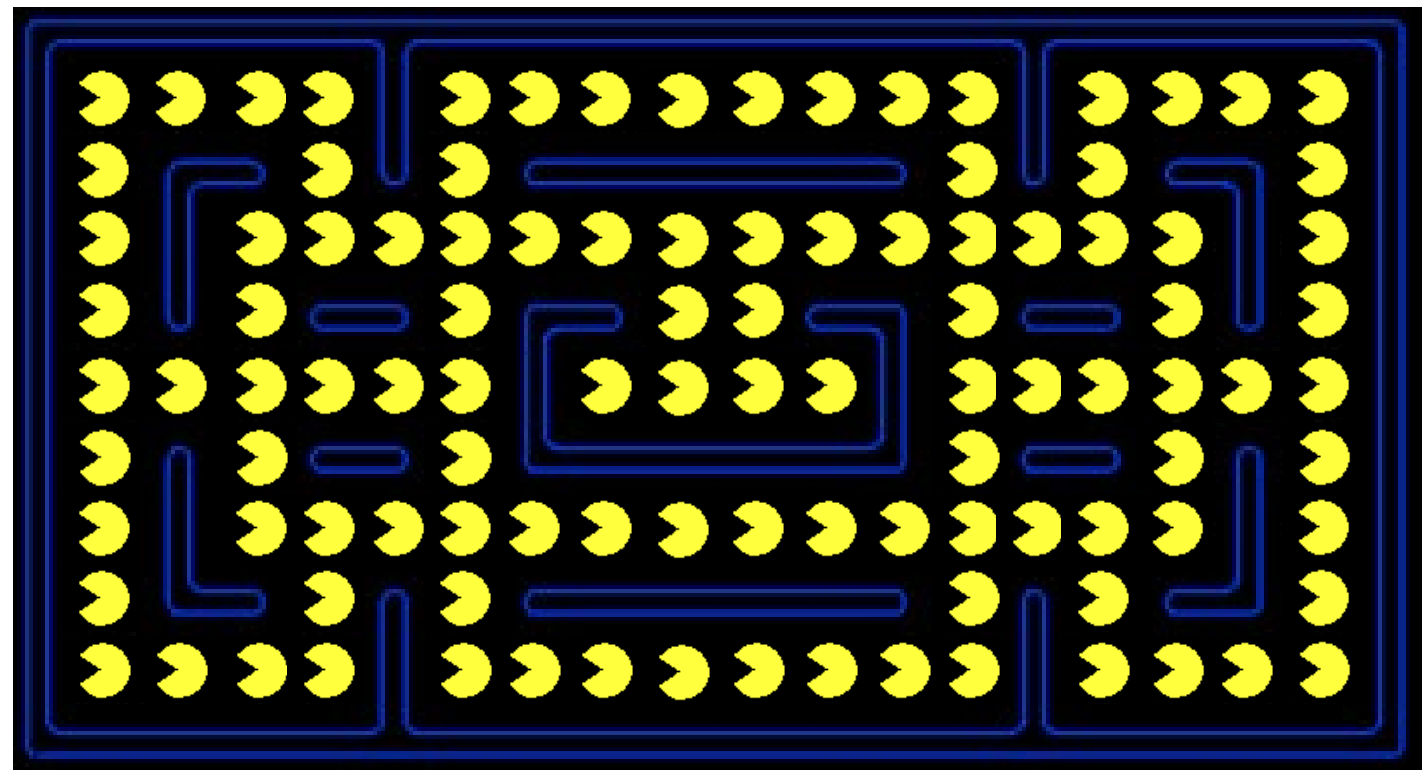
- La **estimación del estado** significa hacer un seguimiento de lo que es cierto ahora, dada una historia de observaciones y acciones.
- Un agente lógico puede preguntarse a sí mismo
 - P. ej., preguntar si $KB \wedge \langle \text{acciones} \rangle \wedge \langle \text{percepciones} \rangle \models At_{2,2_6}$
- Esto es “perezoso”: analiza todo el historial vital a cada paso
- Una forma más “ansiosa” de estimación del estado:
 - Después de cada acción_{t-1} y percepción_{t-1}
 - Para cada variable de estado X_t
 - Si $KB \wedge \text{acción}_{t-1} \wedge \text{percepción}_t \models X_t$, añada X_t a KB
 - Si $KB \wedge \text{acción}_{t-1} \wedge \text{percepción}_t \models \neg X_t$, añada $\neg X_t$ a KB
 - Podemos usar X_t en los siguientes pasos

Ejemplo: localización en un mapa conocido


- Inicializar la KB con las *PacFísicas(+modelo del sensor)* para T pasos temporales
- Ejecutar el agente Pacman durante T pasos temporales:
 - Después de cada acción y percepción
 - Para cada variable $At_{x,y,t}$
 - Si $KB \wedge accion_{t-1} \wedge percepcion_t \models At_{x,y,t}$, añadir $At_{x,y,t}$ a KB
 - Si $KB \wedge accion_{t-1} \wedge percepcion_t \models \neg At_{x,y,t}$, añadir $\neg At_{x,y,t}$ a KB
 - Elige una acción
- Las **posibles** localizaciones de Pacman son aquellas que no son demostrablemente falsas

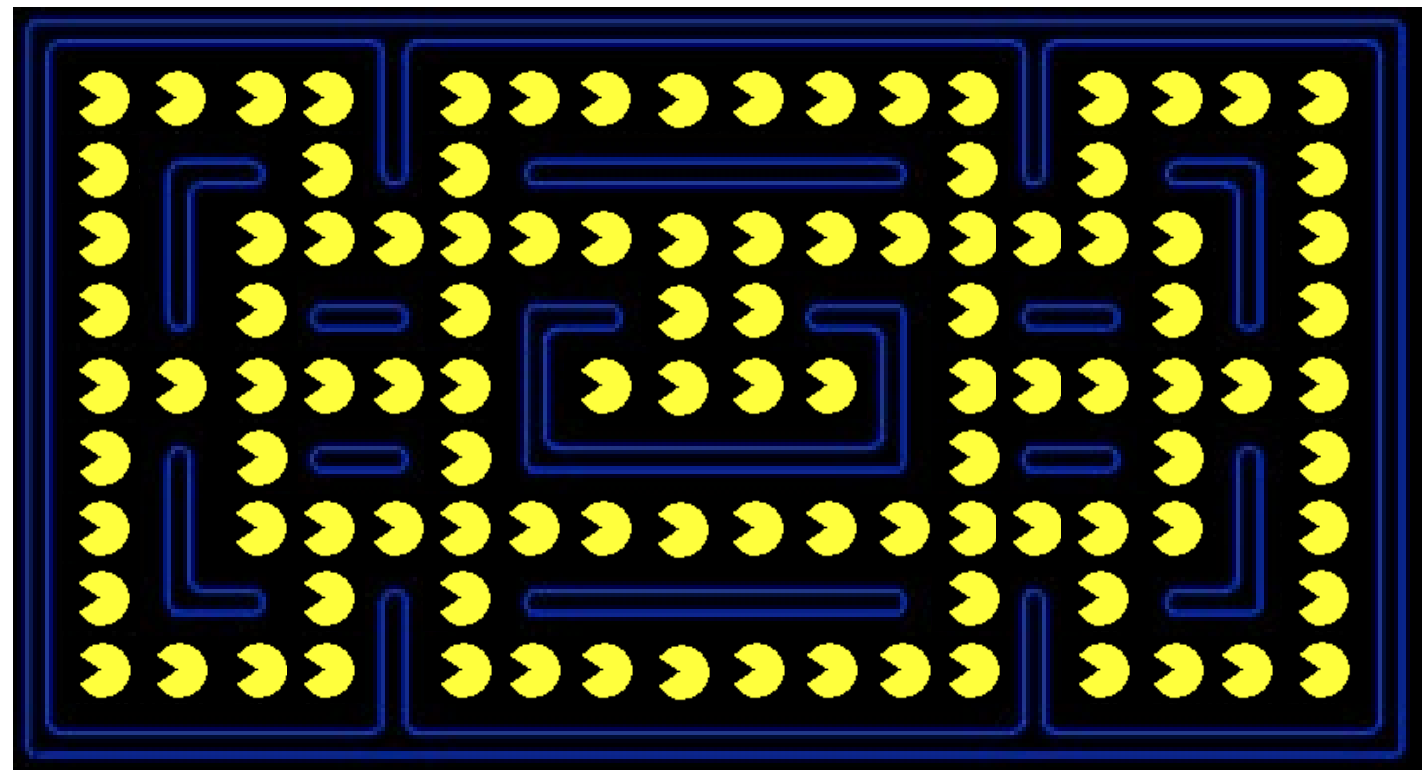
Demo localización

- Percepción 
- Acción
- Percepción
- Acción
- Percepción
- Acción
- Percepción





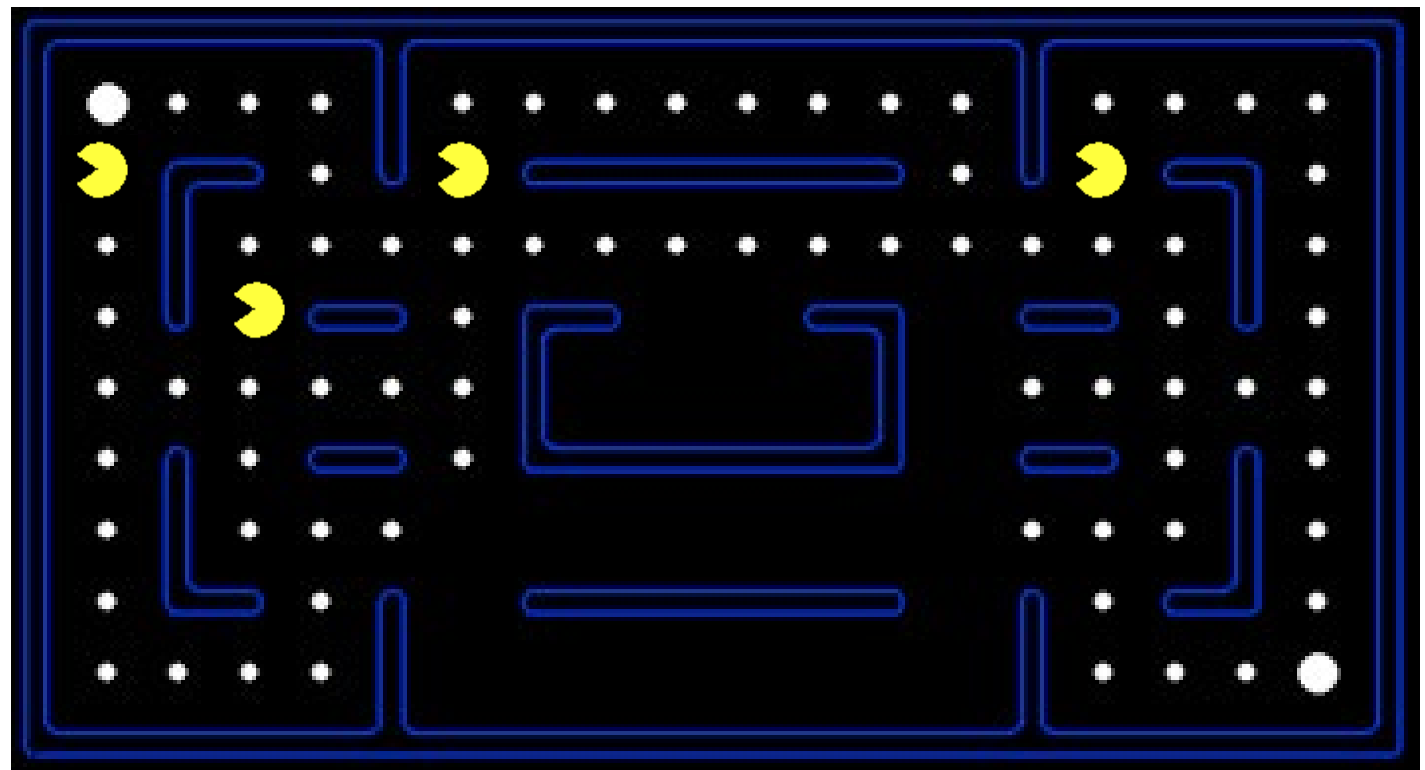
Demo localización

- Percepción 
- Acción *SUR*
- Percepción
- Acción
- Percepción
- Acción
- Percepción






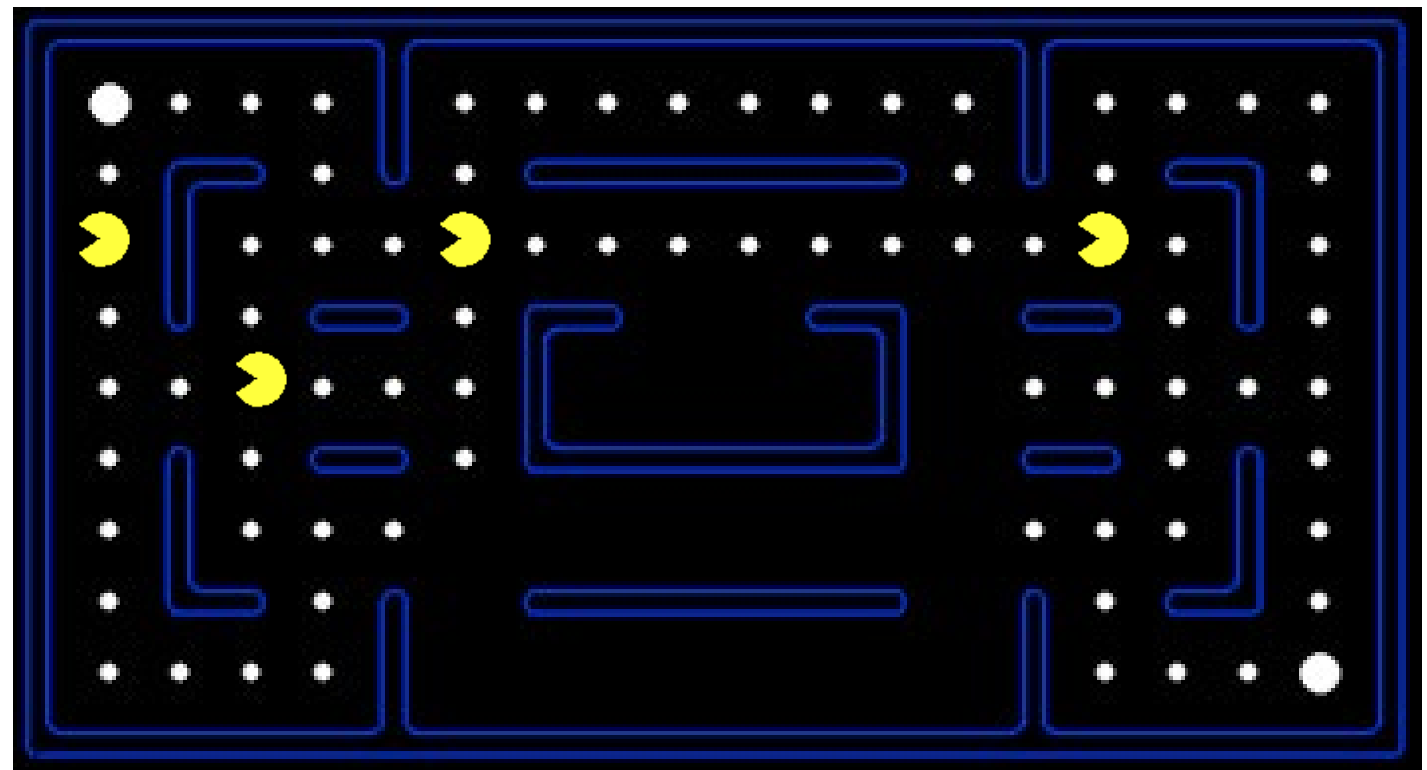
Demo localización

- Percepción 
- Acción *SUR*
- Percepción 
- Acción *SUR*
- Percepción
- Acción
- Percepción




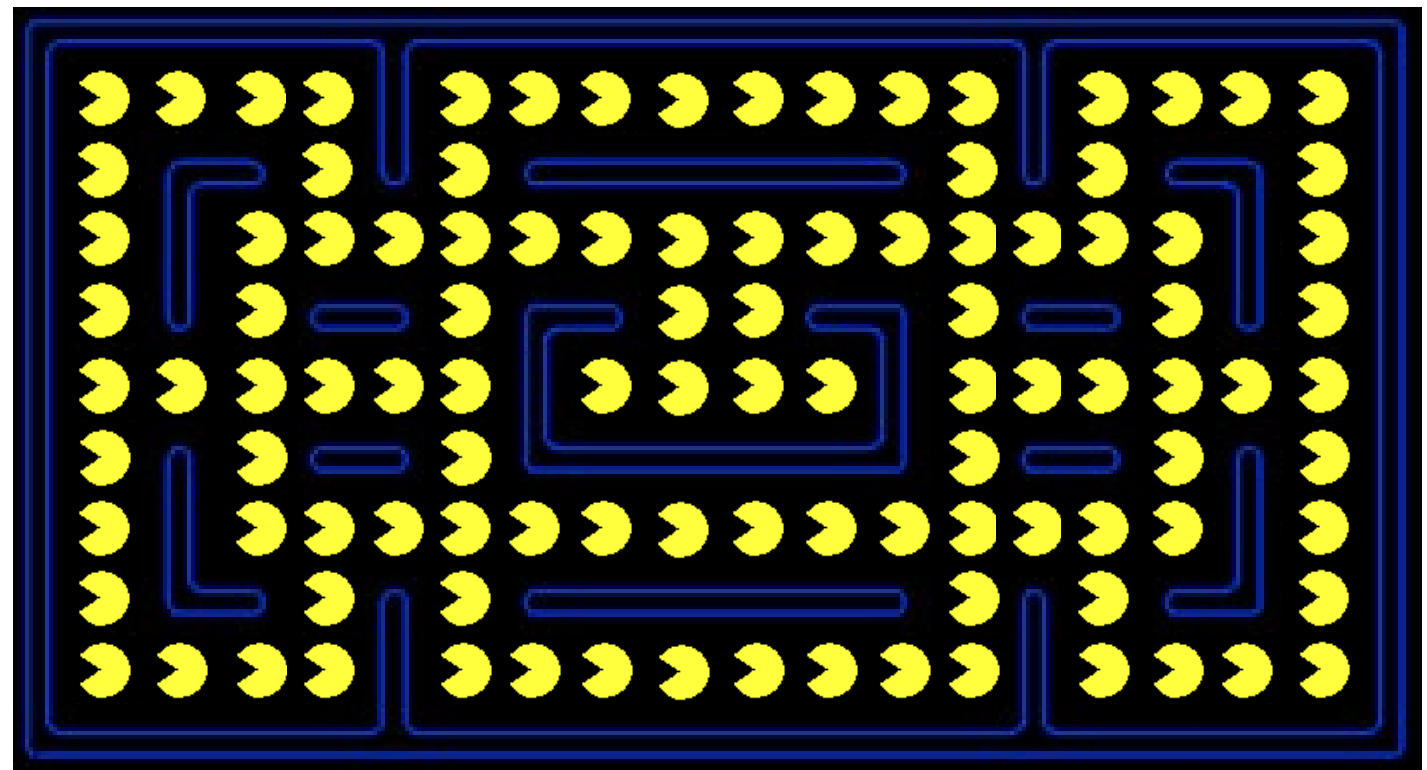
Demo localización

- Percepción 
- Acción *SUR*
- Percepción 
- Acción *SUR*
- Percepción 
- Acción
- Percepción




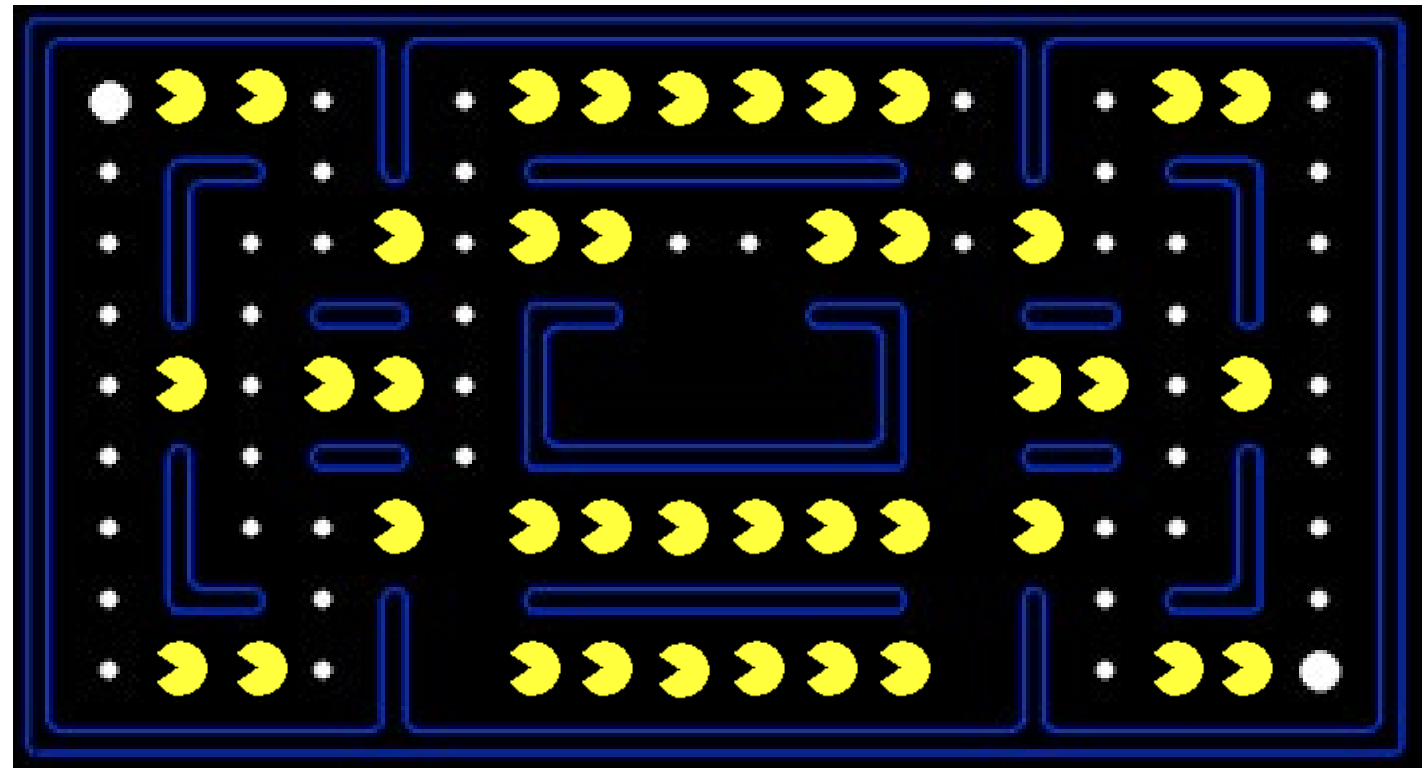
Demo localización

- Percepción 
- Acción
- Percepción
- Acción
- Percepción
- Acción
- Percepción





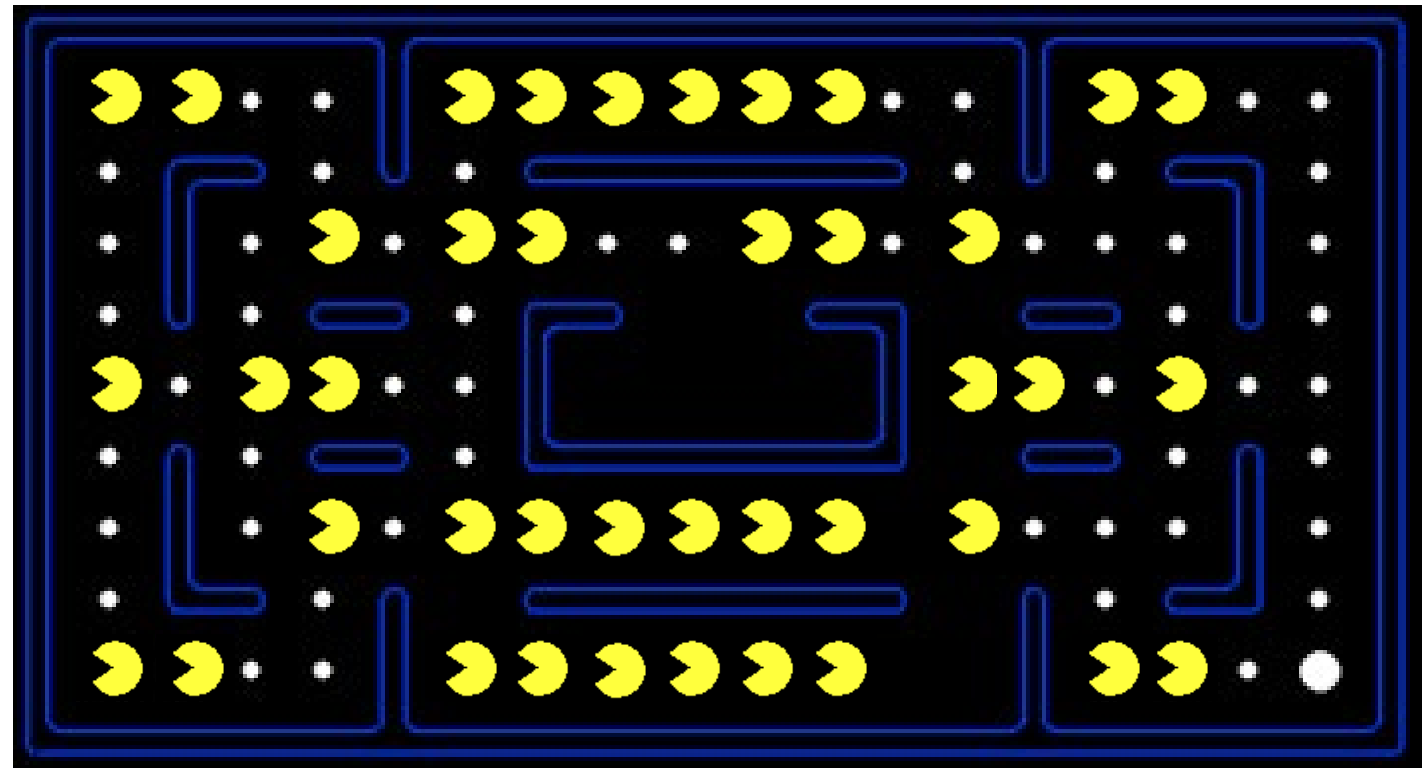
Demo localización

- Percepción 
- Acción *OESTE*
- Percepción
- Acción
- Percepción
- Acción
- Percepción





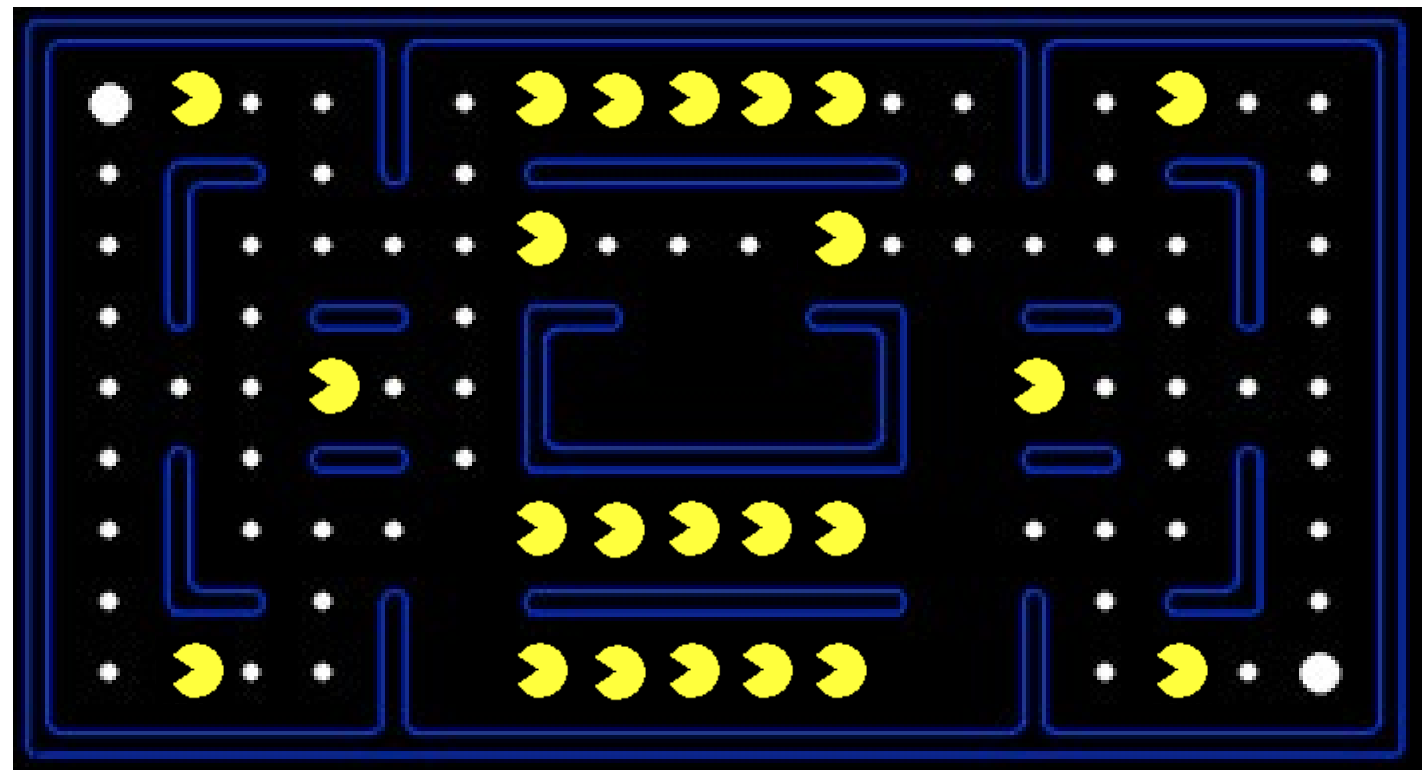
Demo localización

- Percepción 
- Acción *OESTE*
- Percepción 
- Acción
- Percepción
- Acción
- Percepción






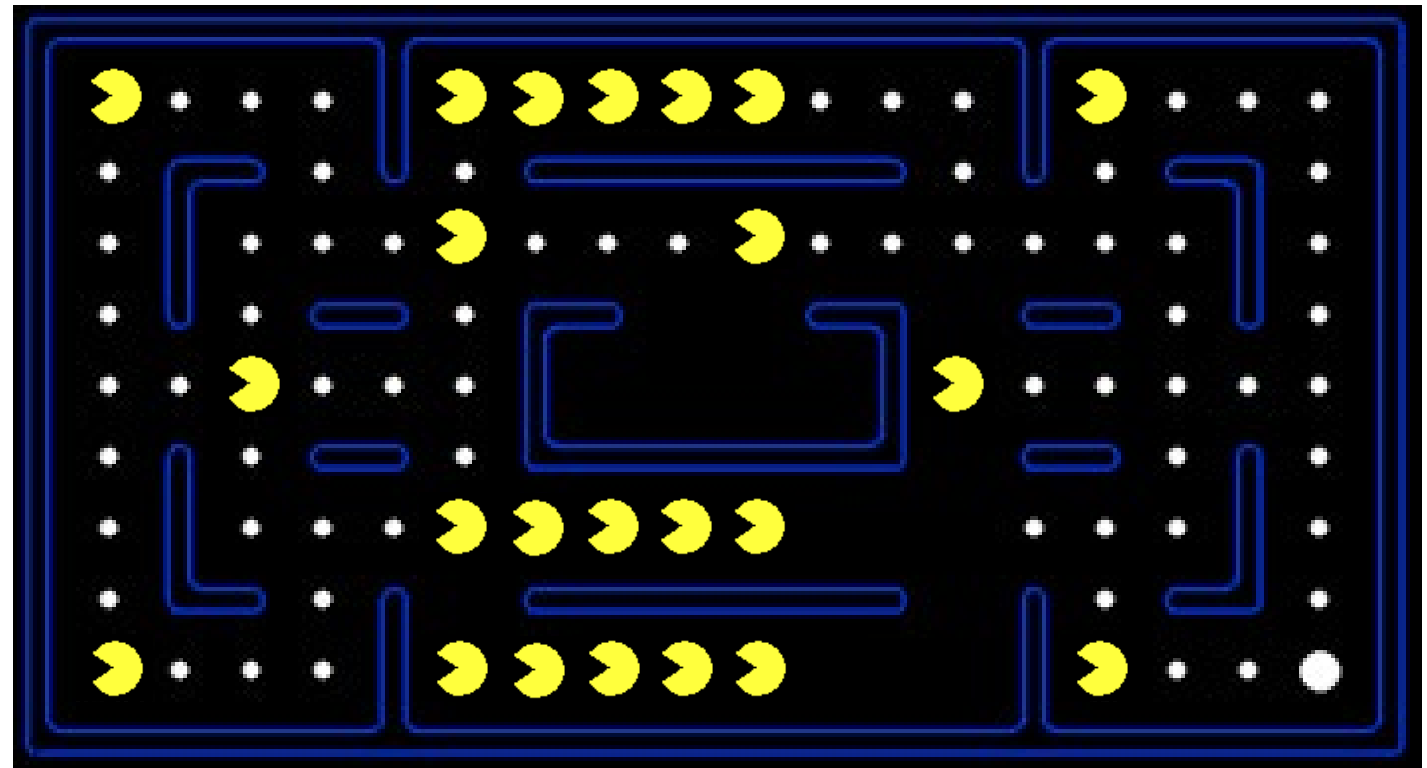
Demo localización

- Percepción 
- Acción *OESTE*
- Percepción 
- Acción *OESTE*
- Percepción
- Acción
- Percepción






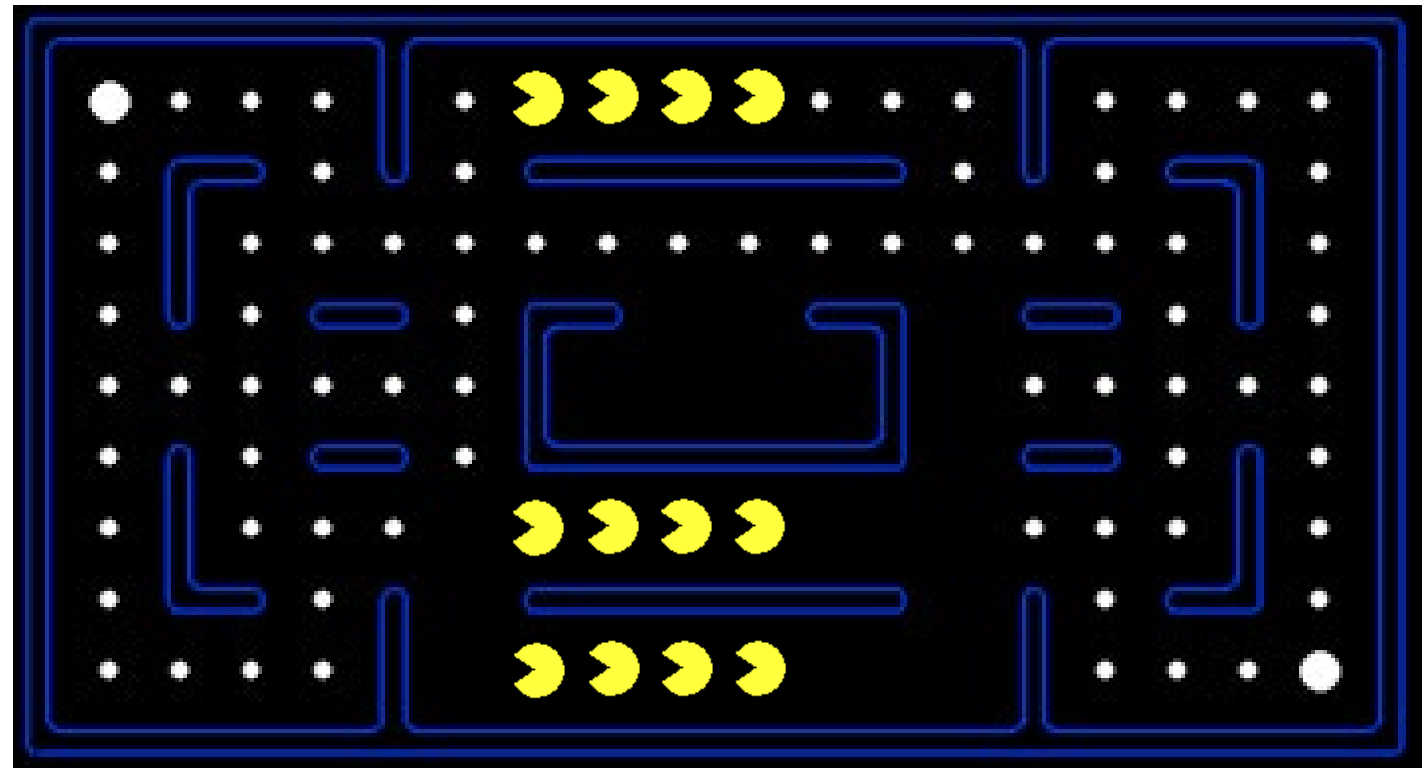
Demo localización

- Percepción 
- Acción *OESTE*
- Percepción 
- Acción *OESTE*
- Percepción 
- Acción
- Percepción





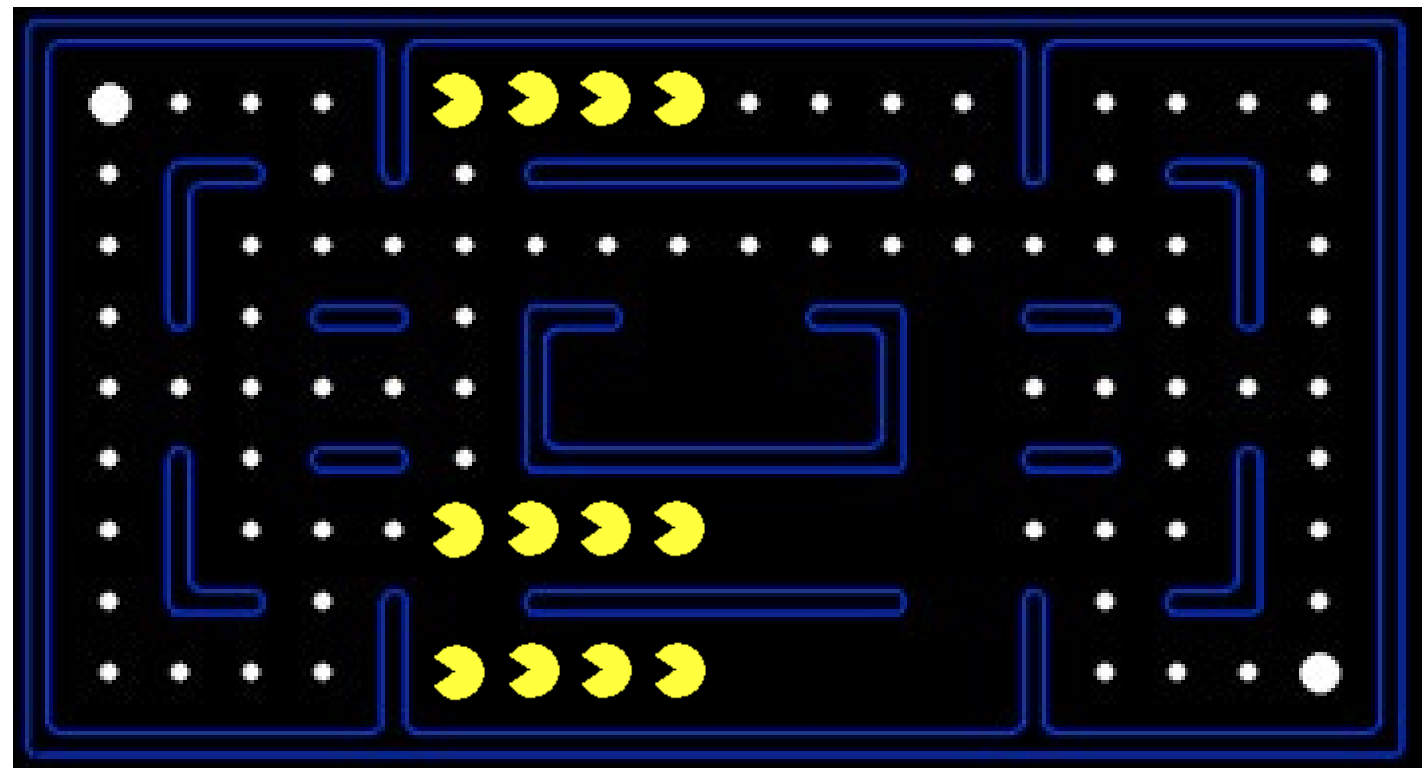
Demo localización

- Percepción 
- Acción *OESTE*
- Percepción 
- Acción *OESTE*
- Percepción 
- Acción *OESTE*
- Percepción

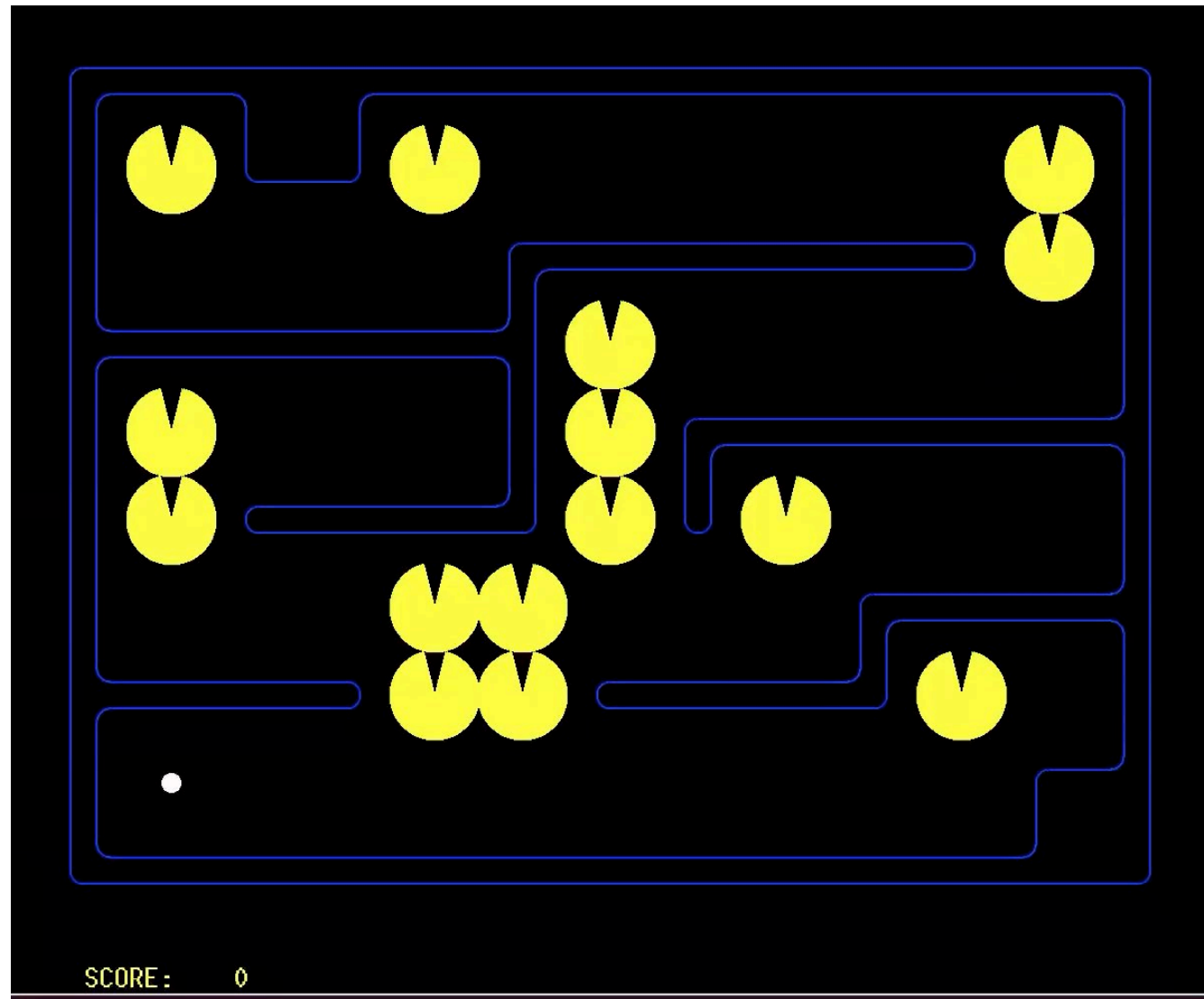


Demo localización

- Percepción 
- Acción *OESTE*
- Percepción 
- Acción *OESTE*
- Percepción 
- Acción *OESTE*
- Percepción 



Localización con movimiento aleatorio



Ejemplo: mapa desde una pos. relativa conocida

- Sin pérdida de generalidad, llamemos a la posición inicial 0,0
- Las percepciones **Blocked** le dicen a Pacman qué acciones funcionan, para que siempre estime dónde está:
 - “*Dead reckoning*” (navegación por estima)
- Inicializar la KB con las **PacFísicas** para T pasos temporales, empezando en 0,0
- Ejecutar el agente Pacman durante T pasos temporales
 - En cada paso temporal
 - Actualizar la KB con los hechos de acción previos y los de percepción nuevos
 - Para cada variable de pared **Wall_{x,y}**
 - Si **Wall_{x,y}** viene implicada, añadir a KB
 - Si \neg **Wall_{x,y}** viene implicada, añadir a KB
 - Elegir una acción
- Las variables de pared constituyen el mapa

Demo de mapeado

- Percepción 
- Acción *NORTE*
- Percepción 
- Acción *ESTE*
- Percepción 
- Acción *SUR*
- Percepción 



Simultaneous localization and mapping: SLAM

- A menudo, el *dead reckoning* no funciona en el mundo real
 - P. ej., los sensores solo cuentan el número de paredes adyacentes (0,1,2,3 = 2 bits)
- Pacman no sabe qué acciones funcionan, por lo que se “pierde”
 - Si no sabe dónde está, ¿cómo va a hacer un mapa?
- Inicializar la KB con las *PacFísicas* para T pasos temporales, empezando en 0,0
- Ejecuta el agente Pacman durante T pasos temporales.
 - En cada paso temporal:
 - Actualizar la KB con los hechos de acción previos y los de percepción nuevos
 - Para cada x,y , añadir o bien $\text{Wall}_{x,y}$ o bien $\neg\text{Wall}_{x,y}$ a la KB, si viene implicado
 - Para cada x,y , añadir o bien $\text{At}_{x,y,t}$ o bien $\neg\text{At}_{x,y,t}$ a la KB, si viene implicado
 - Elige una acción

Inferencia (recordatorio)

- Método 1: *comprobación de modelos*
 - Para cada mundo posible, si α es verdad, asegura que β también lo sea
- Método 2: *demostración de teoremas*
 - Búsqueda de una secuencia de pasos de prueba (aplicación de las reglas de inferencia) que lleven de α a β
- Algoritmo **sólido**: deriva solo sentencias implicadas (no se inventa cosas)
- Algoritmo **completo**: puede derivar cualquier sentencia que esté implicada

Demostración de teoremas (básica)

- El **encadenamiento hacia delante (FC)** aplica el *Modus Ponens* para generar nuevos hechos:
 - *Dados* $X_1 \wedge X_2 \wedge \dots \wedge X_n \Rightarrow Y$ y X_1, X_2, \dots, X_n , *inferir* Y
- El encadenamiento hacia delante sigue aplicando esta regla, añadiendo nuevos hechos, hasta que ya no se puede añadir nada más
- Requiere que la KB contenga solo **clausulas definidas**:
 - (Conjunción de símbolos) \Rightarrow símbolo; o
 - Un solo símbolo (nótese que X es equivalente a $\text{Verdadero} \Rightarrow X$)

Ejemplo de encadenamiento hacia delante (FC)

- $A \Rightarrow B$
- $A \Rightarrow C$
- $B \wedge C \Rightarrow D$
- $D \wedge E \Rightarrow Q$
- $A \wedge D \Rightarrow Q$
- A
- $¿Q?$

Propiedades del encadenamiento hacia delante

- Teorema: FC es **sólido** y **completo** para KB formadas por cláusulas definidas
- Se ejecuta en tiempo **lineal** (si se implementa con algunos trucos de indexación)
- Utilizado en sistemas reales: p.ej., para actualizar bases de datos

Encadenamiento hacia atrás (brevemente)

- El **encadenamiento hacia atrás** (BC) funciona al revés:
 - Empezar desde el objetivo Y a probar
 - Encontrar implicaciones $X_1 \wedge X_2 \wedge \dots \wedge X_n \Rightarrow Y$ con Y en el lado derecho en la KB
 - Fijar X_1, X_2, \dots, X_n como subobjetivos para demostrar recursivamente (parar en hechos conocidos)
- Teorema: BC es **sólido** y **completo** para KB de cláusulas definidas
- Tiempo **lineal** (si se maneja inteligentemente la indexación y el almacenamiento en caché)

Ejemplo de encadenamiento hacia atrás (BC)

- $A \Rightarrow B$
- $A \Rightarrow C$
- $B \wedge C \Rightarrow D$
- $D \wedge E \Rightarrow Q$
- $A \wedge D \Rightarrow Q$
- A
- $¿Q?$

Resolución (brevemente)

- Cada clausula CNF puede escribirse como:
 - Conjunción de símbolos \Rightarrow disyunción de símbolos
- La regla de inferencia de resolución toma dos clausulas e infiere una nueva clausula **resolviendo** símbolos complementarios (en lados opuestos de \Rightarrow):

- Ejemplo: $A \wedge B \wedge C \Rightarrow U \vee V$

$$D \wedge E \wedge U \Rightarrow X \vee Y$$

$$A \wedge B \wedge C \wedge D \wedge E \Rightarrow V \vee X \vee Y$$

- La resolución es **completa** para la lógica proposicional
- Tiempo exponencial en el peor de los casos

Resumen

- La inferencia lógica calcula relaciones de implicación entre sentencias
- Los demostradores de teoremas aplican reglas de inferencia a las sentencias:
 - El encadenamiento hacia delante aplica modus ponens con cláusulas definidas; tiempo lineal
 - La resolución es completa para lógica proposicional, pero el tiempo es exponencial en el peor de los casos
- Los solucionadores SAT basados en DPLL proporcionan una inferencia increíblemente eficiente.
- Los agentes lógicos pueden hacer localización, mapeo, SLAM, planificación (y muchas otras cosas) utilizando un algoritmo de inferencia genérico sobre una base de conocimiento.