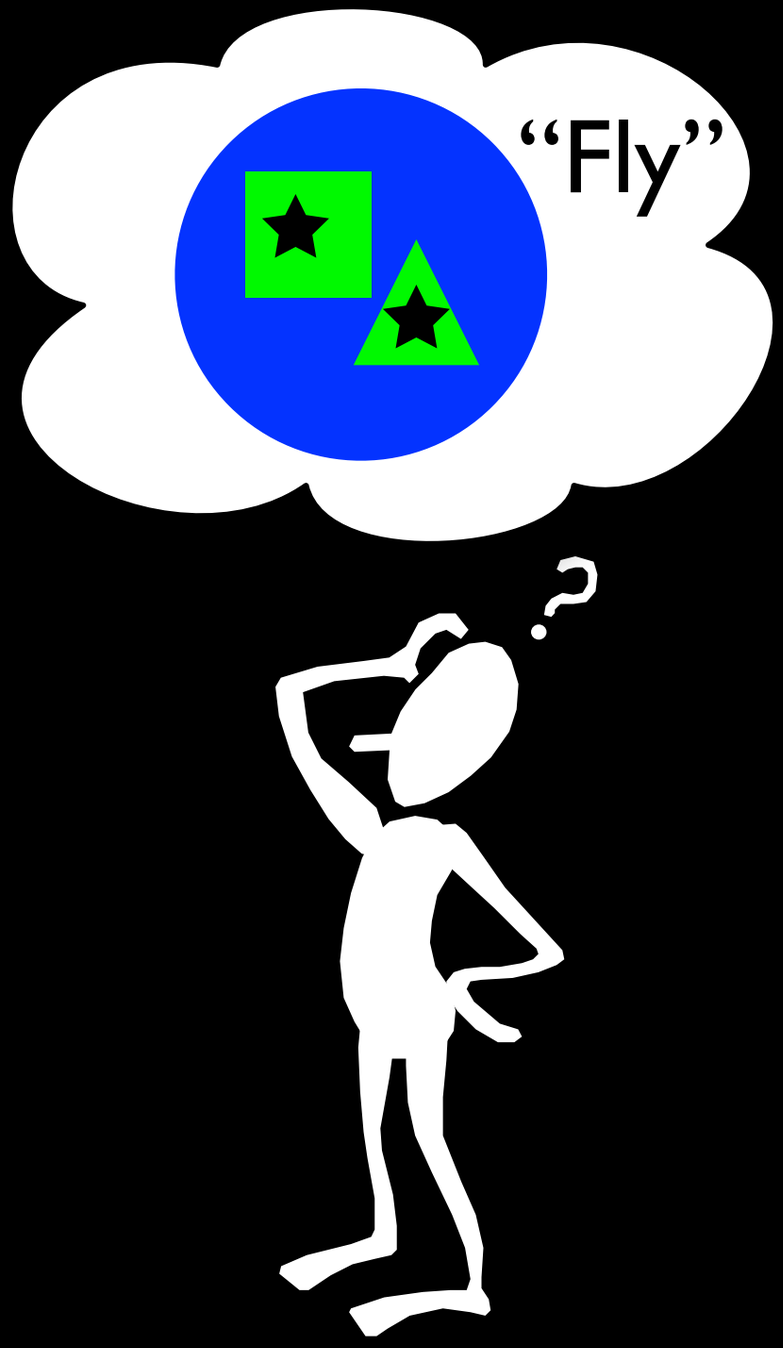
CSC242: Intro to AI

Lecture 8

LOGIC

Propositional Logic

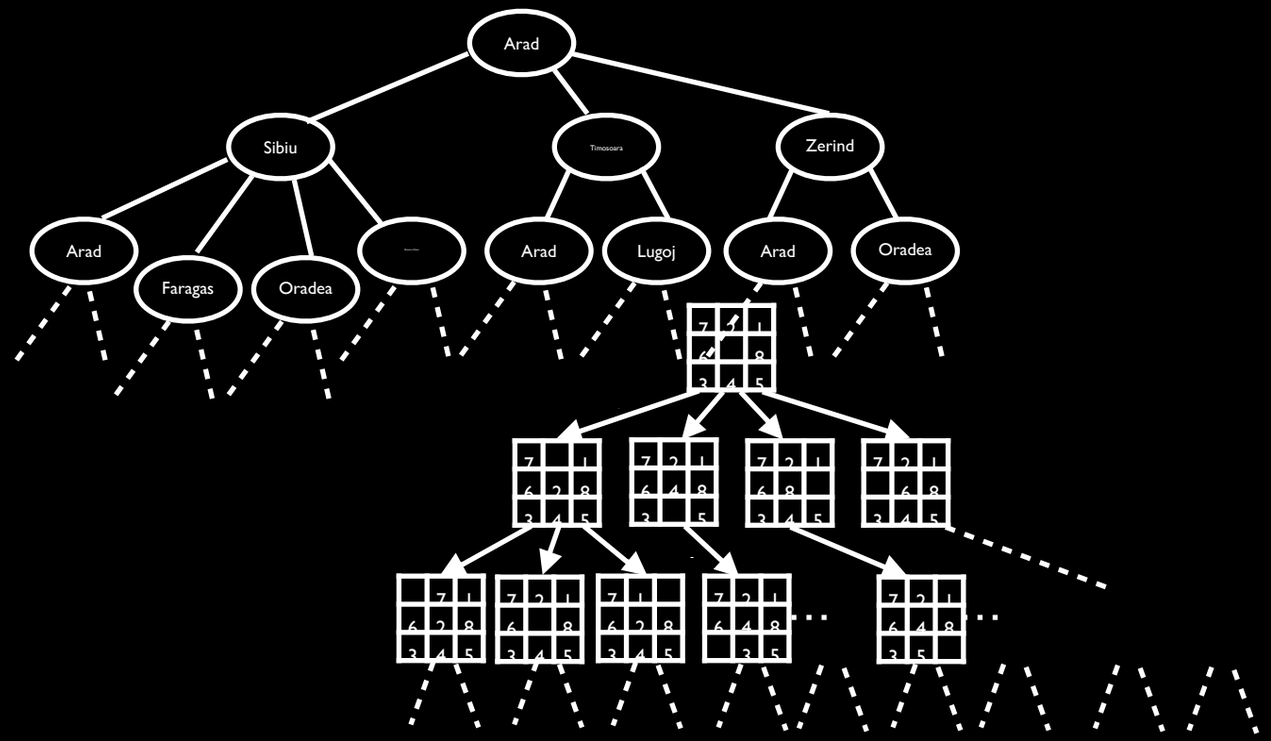


Problem-solving state

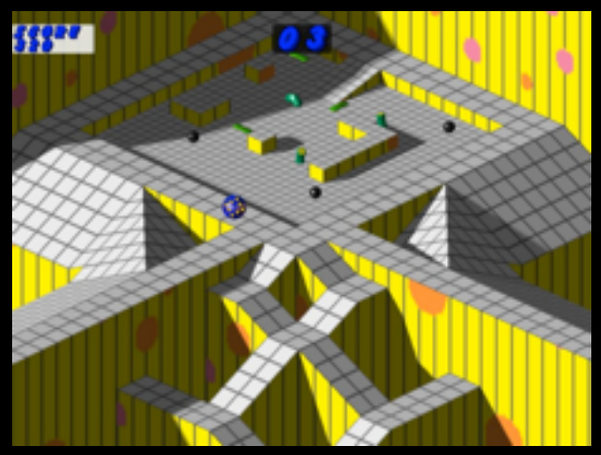
Transition Model

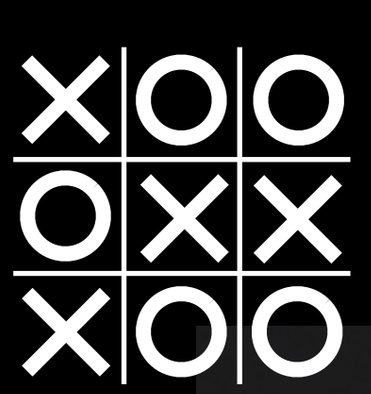
World state

Action


Arad
Sibiu
Timosoara
Zerind
Arad
Faragas
Oradea
Arad
Lugoj
Arad
Oradea


Northern Territory
Queensland
Western Australia
South Australia
New South Wales
Victoria
Tasmania


03




1
2
3
4
5
6
a b c d e f g h

LOGIC

- A formal language for representing knowledge
- What makes a language a language?
  - Can compose pieces together: syntax
  - It has a meaning: semantics
- What makes logic a “formal” language?
  - Simple, unambiguous syntax
  - Simple, unambiguous semantics

The Problem with English

I saw a man on the hill with a telescope

Hunt The Wumpus

```
Hunt the Wumpus
Original BASIC version (1972) by Gregory Yob.
Inform port (1999) by Magnus Olsson <zebulon@pobox.com>.
Release 1 / Serial number 991216 / Inform v6.21

Type 1 to read the instructions, 2 to read the implementation n
the game, or 4 to quit.
3

You can choose between the following caves:
1: The Dodecahedron
2: The Möbius Strip
3: The String of Beads
4: The Dendrite
5: The One-way Lattice
Which cave (1-5)? 1
OK, using the Dodecahedron

Bats nearby!
You are in room 1
Tunnels lead to 2 5 8
Shoot, Move or Quit (S-M-Q)?
```

C:\Documents and Settings\jatwood\Desktop\Wumpus_vsnet_solution\HuntTheWump...

```
*** Wumpus .NET ***

Based on:

                          WUMPUS 2
                    CREATIVE COMPUTING
                  MORRISTOWN  NEW JERSEY



INSTRUCTIONSn
CAVE #(0-6) 1

HUNT THE WUMPUS

YOU ARE IN ROOM 13
TUNNELS LEAD TO 11 14 15

SHOOT OR MOVE m
WHERE TO 14


I FEEL A DRAFT!
YOU ARE IN ROOM 14
TUNNELS LEAD TO 12 13 16

SHOOT OR MOVE m
WHERE TO 13


YOU ARE IN ROOM 13
TUNNELS LEAD TO 11 14 15

SHOOT OR MOVE m
WHERE TO 11


I SMELL A WUMPUS!
YOU ARE IN ROOM 11
TUNNELS LEAD TO 12 13 9

SHOOT OR MOVE s
NO. OF ROOMS 1
ROOM #12

MISSED

YOU ARE IN ROOM 11
TUNNELS LEAD TO 12 13 9
```

TEXAS INSTRUMENTS

HOME COMPUTER

HUNT THE WUMPUS

ARCADE ENTERTAINMENT

SOLID STATE CARTRIDGE

This game can be played using the optional Wired Remote Controllers.

An exciting simulated hunt in a hidden maze of caverns and twisting tunnels. Seek out the lair of the Wumpus, while avoiding perils along the way!


Hunt the Wumpus

Hunted Wumpus 3G

Creature — Beast

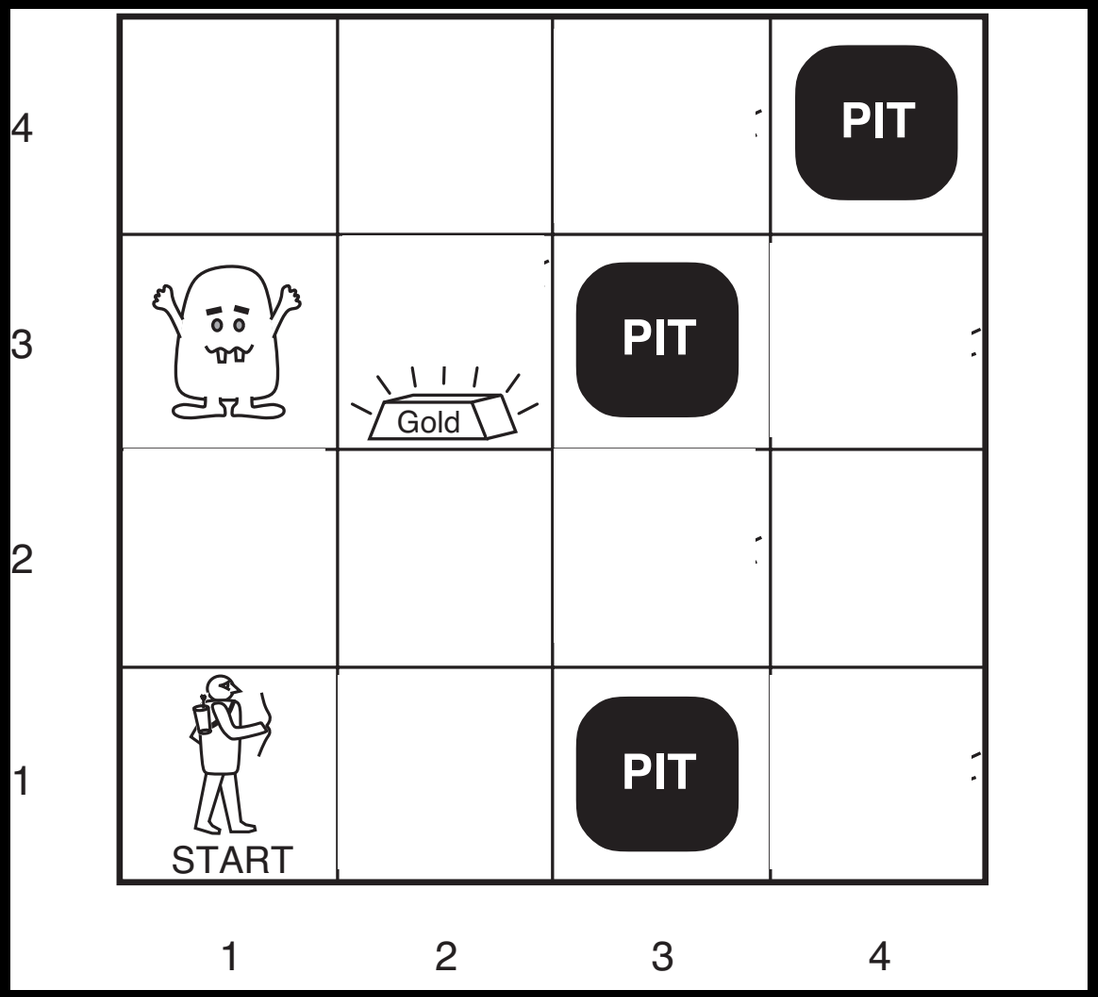
When Hunted Wumpus comes into play, each other player may put a creature card from his or her hand into play.

Just one can feed a dozen people for a month.

6/6

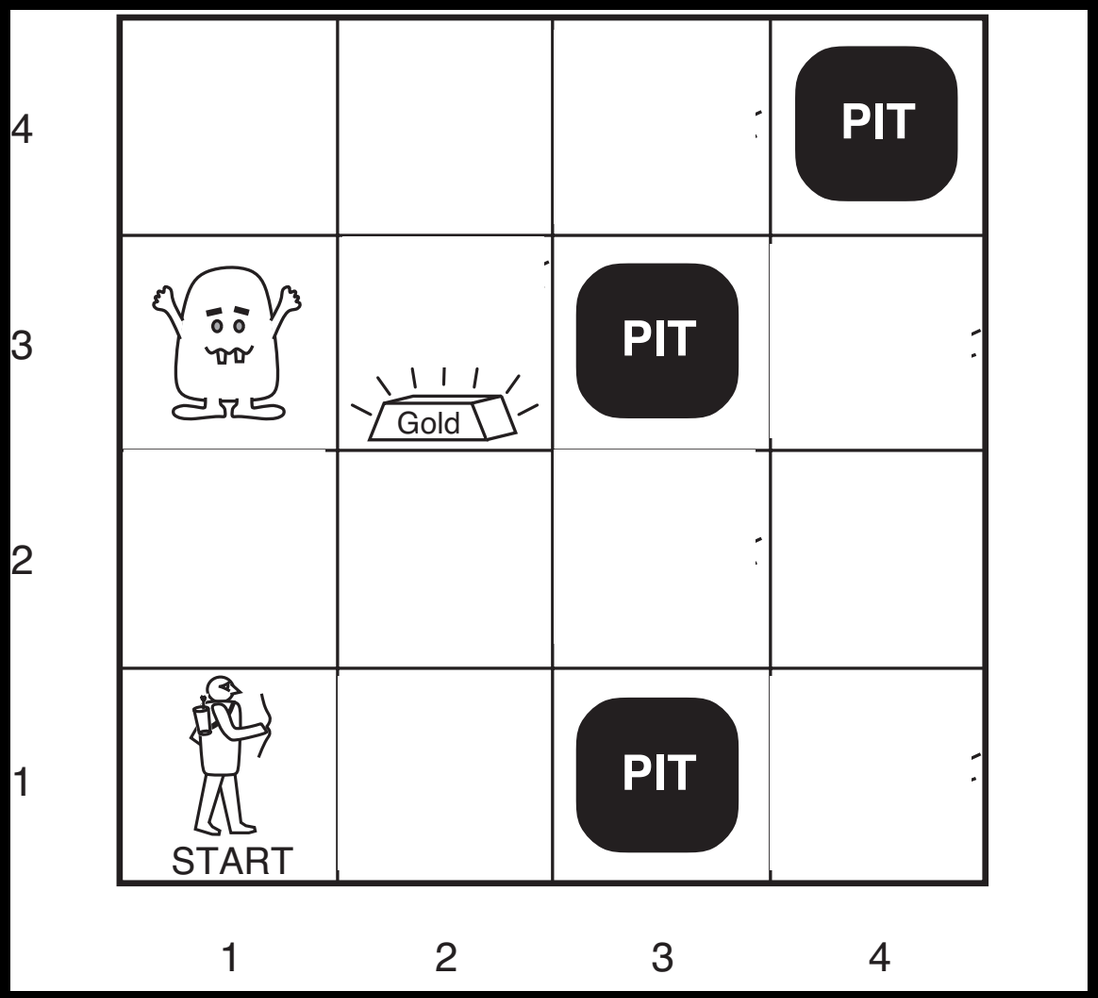
Thomas M. Baxa

Hunt The Wumpus



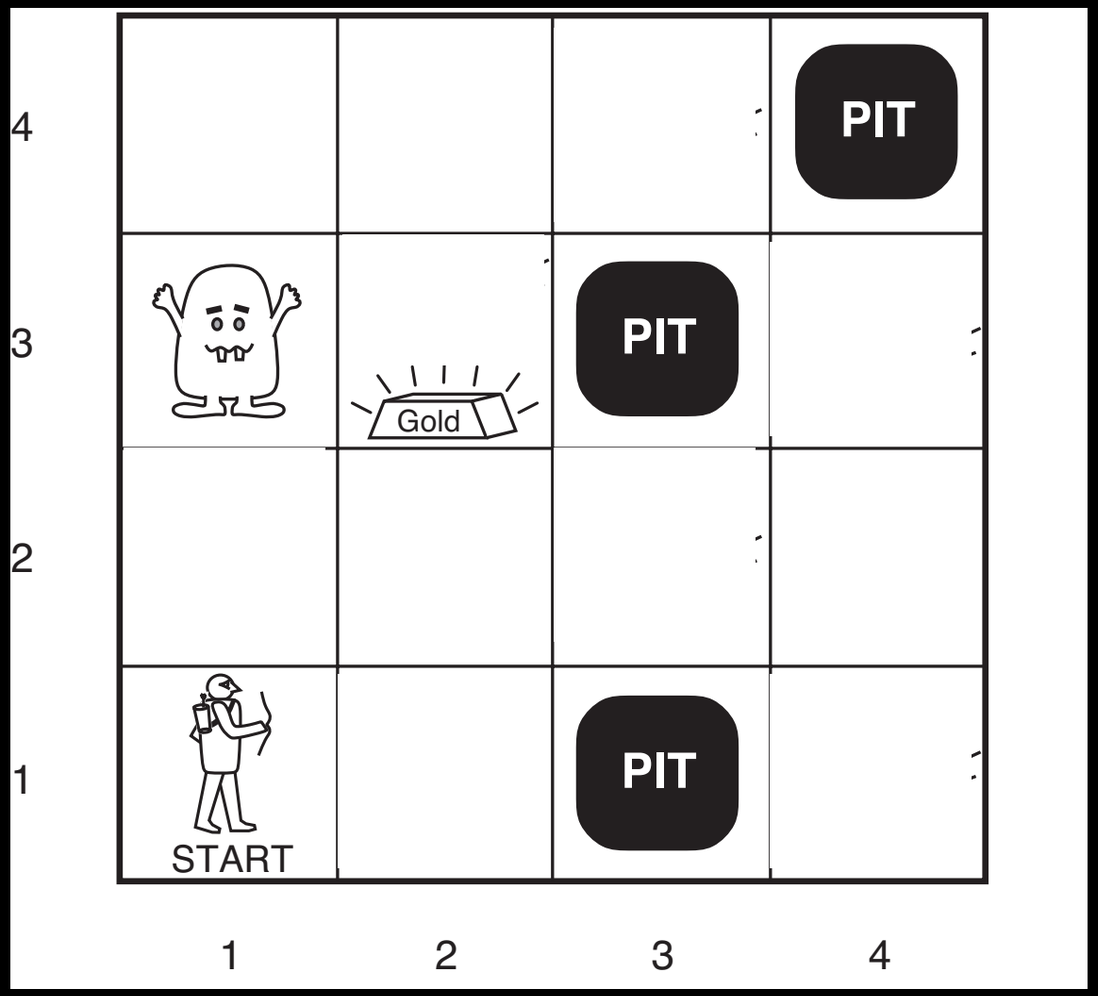
- n x n grid of rooms
- Agent starts in [1,1] facing left
- Gold, wumpus in random square (not [1,1])
- Wumpus doesn't move
- Pits random (P=0.2)

Hunt The Wumpus



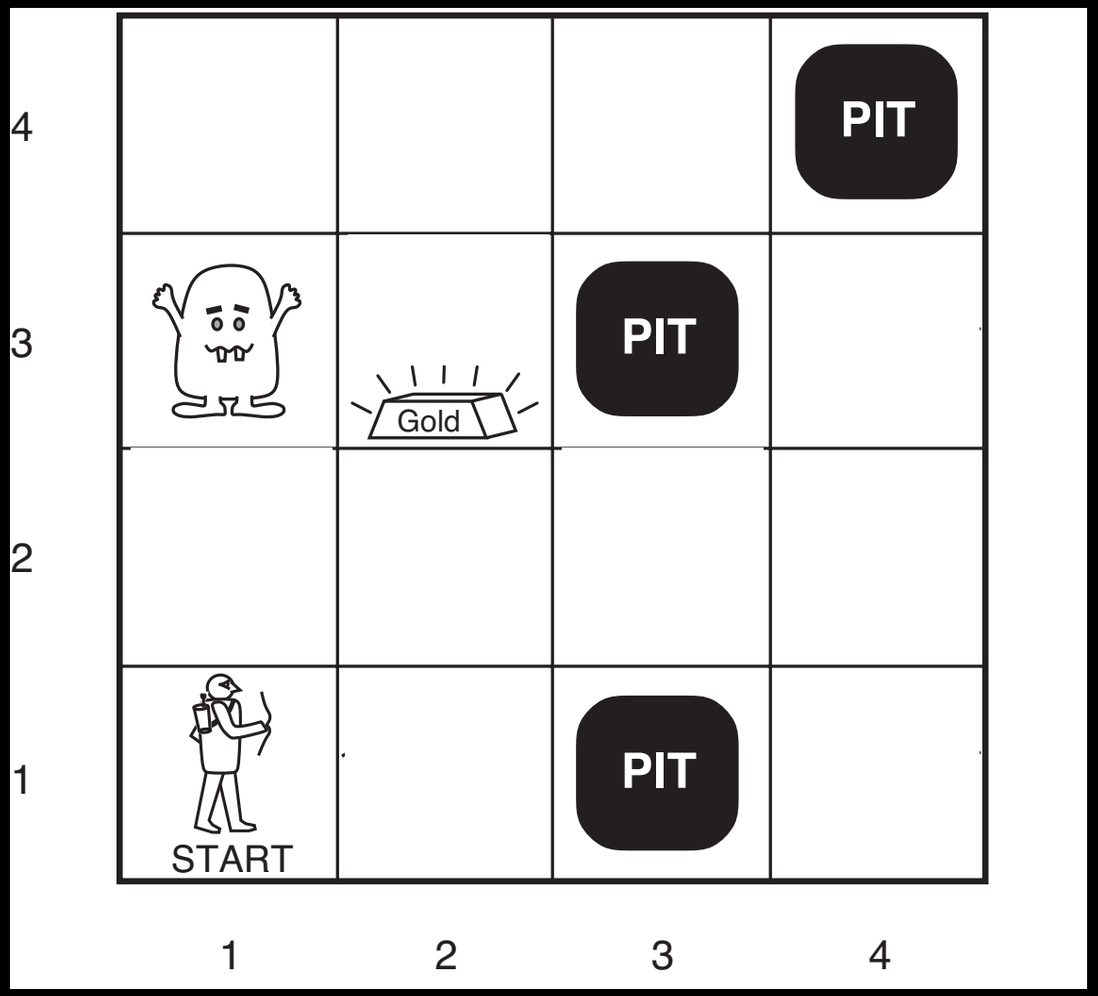
- Move fwd, turn left/right
- Grab gold
- Shoot arrow (once)
- Climb out (from [1,1])
- Costs:
  - -1 per move
  - -10 to shoot

Hunt The Wumpus

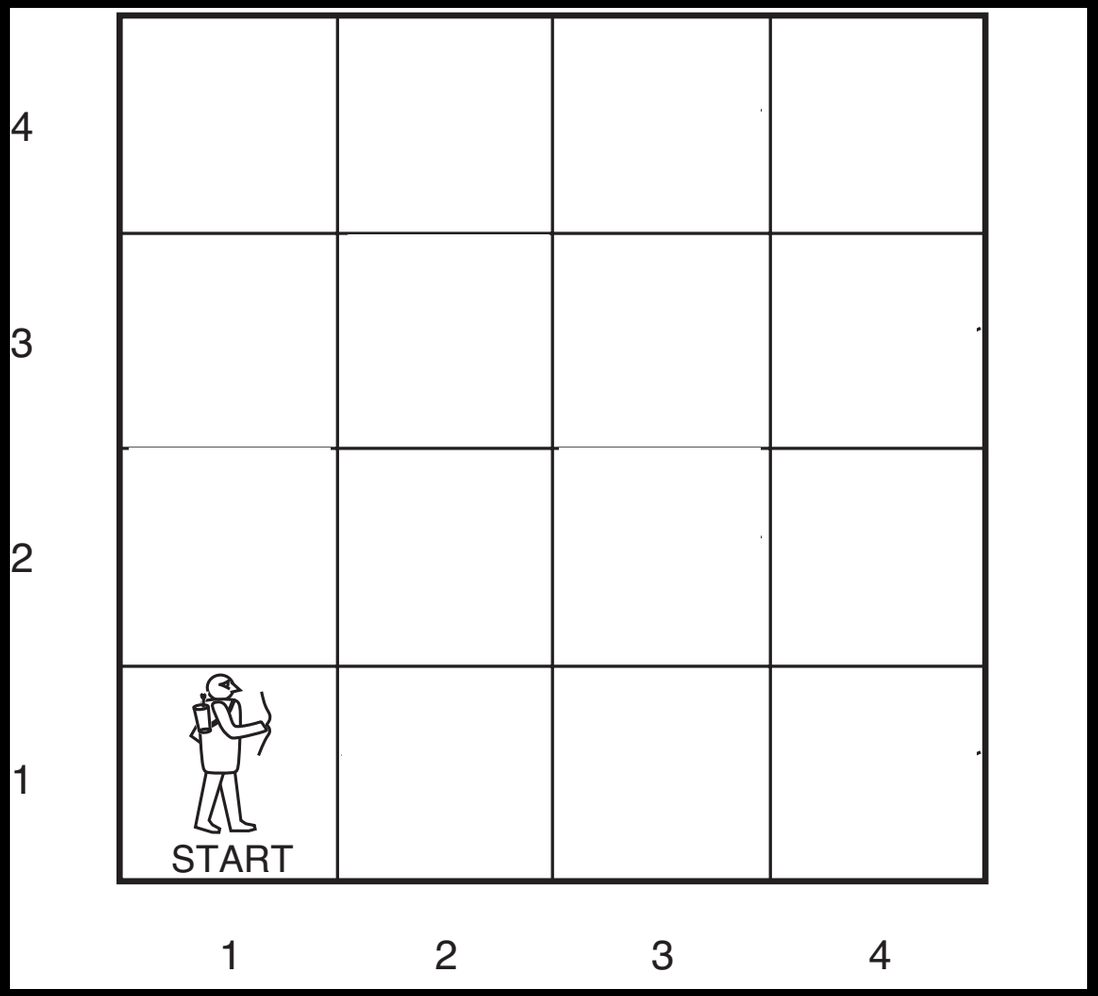


- Get gold and climb out: +1000
- Fall in pit or get eaten by wumpus: -1000

Hunt The Wumpus

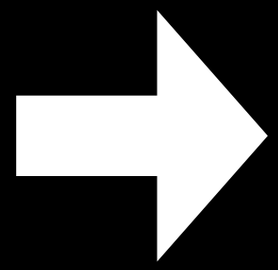


Hunt The Wumpus



Partially Observable

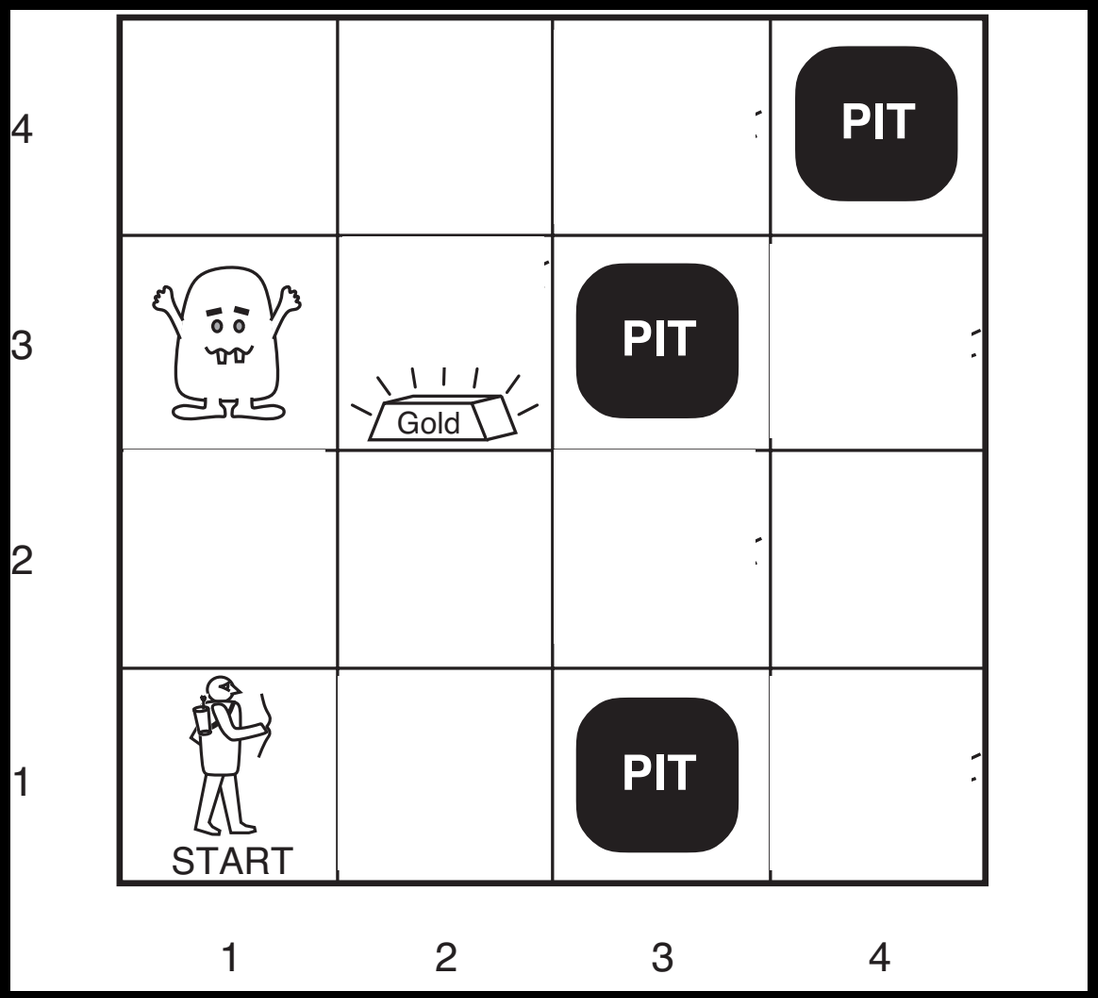
Don't know everything about the state of the world



Have to represent uncertainty in our knowledge

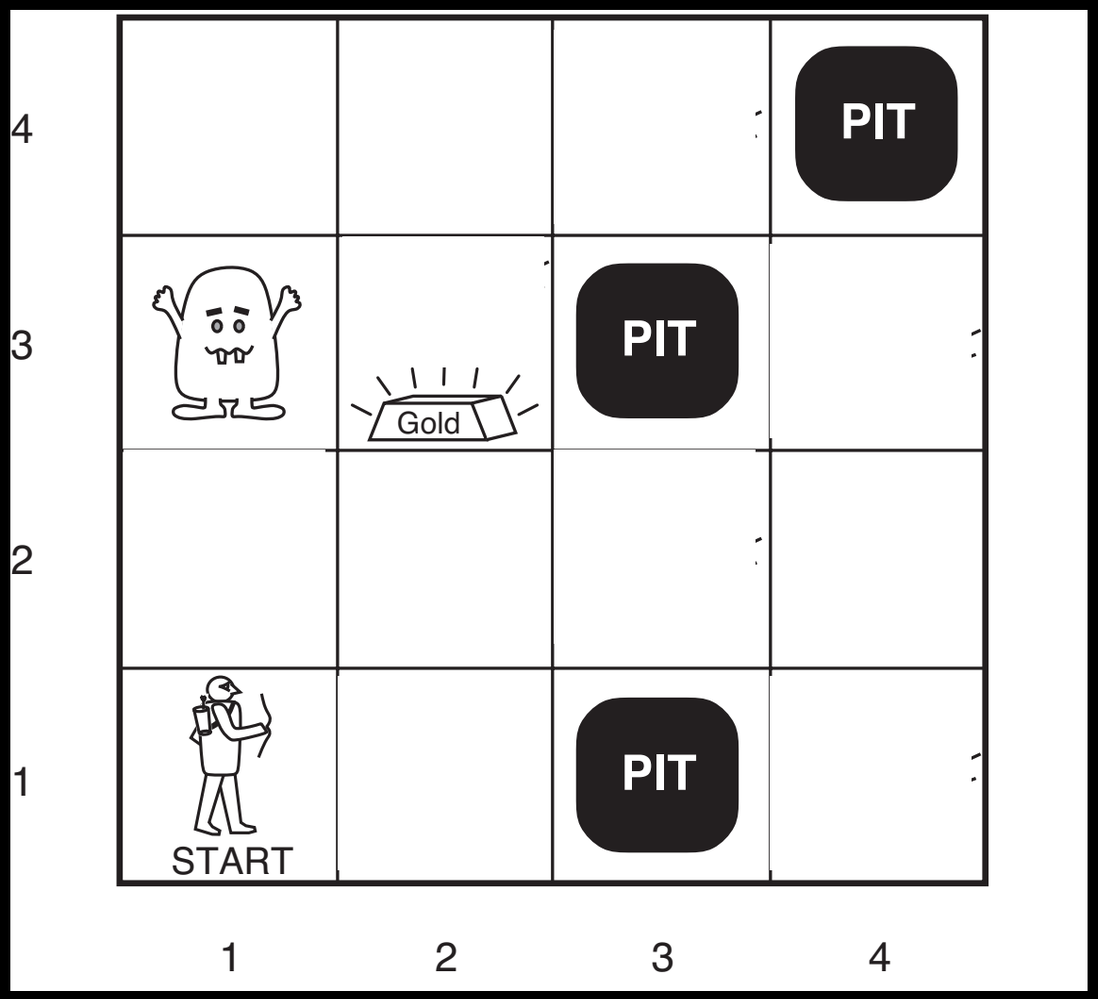
Have to explore

Representation



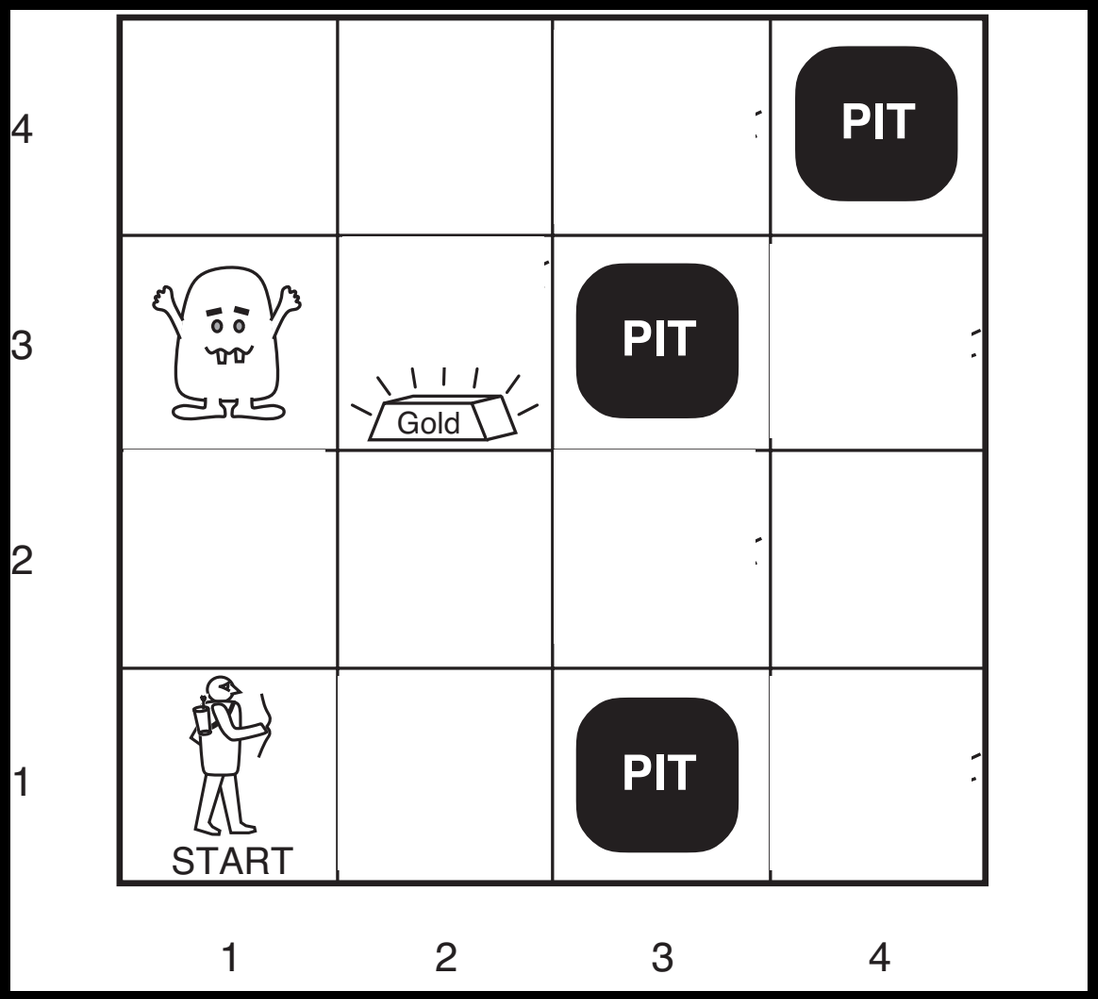

```
For each room:
   Has pit? true or false
   Has gold? true or false
   Has wumpus? true or false
```

Representation



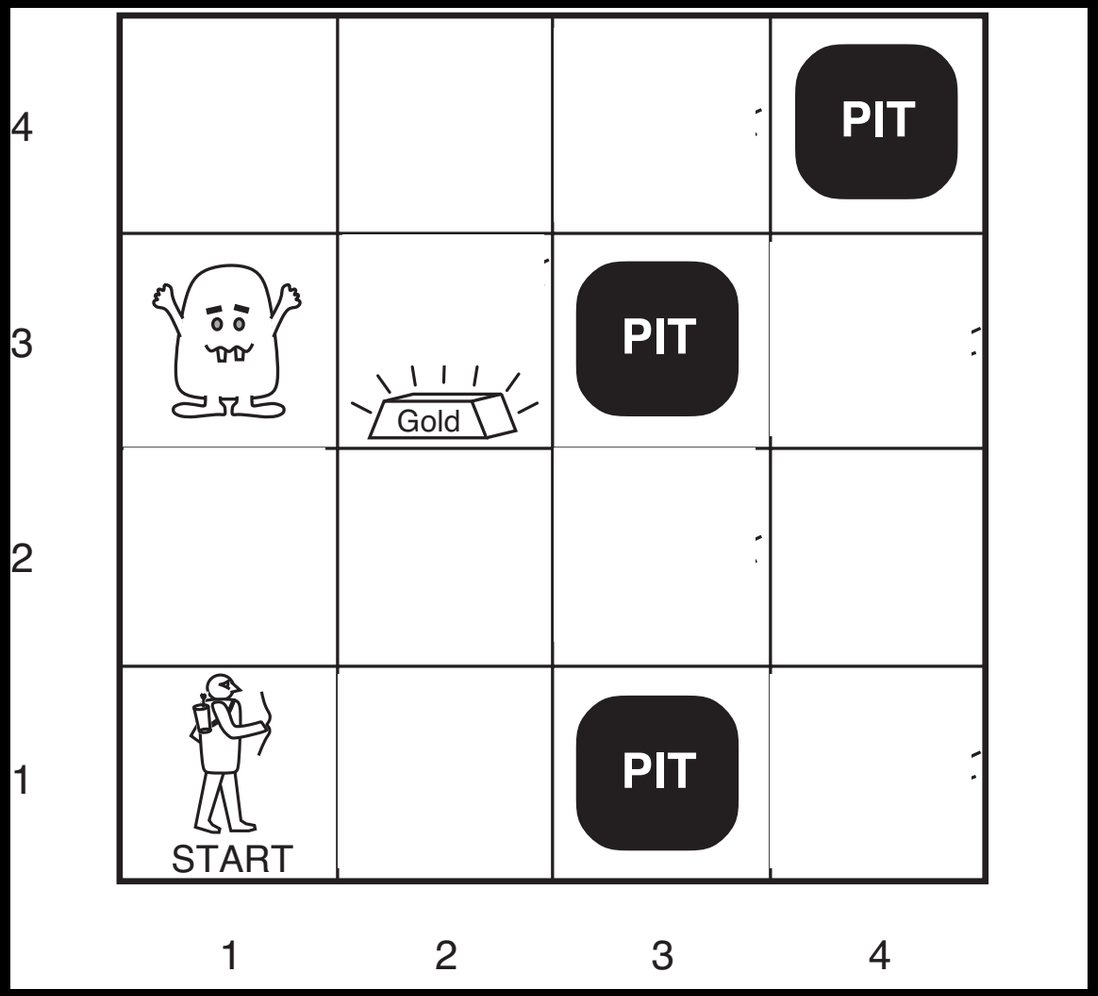

```
Boolean[][][] env =
     new Boolean[n][n][3];

final int PIT = 0;
final int GOLD = 1;
final int WUMPUS = 2;
```

Representation



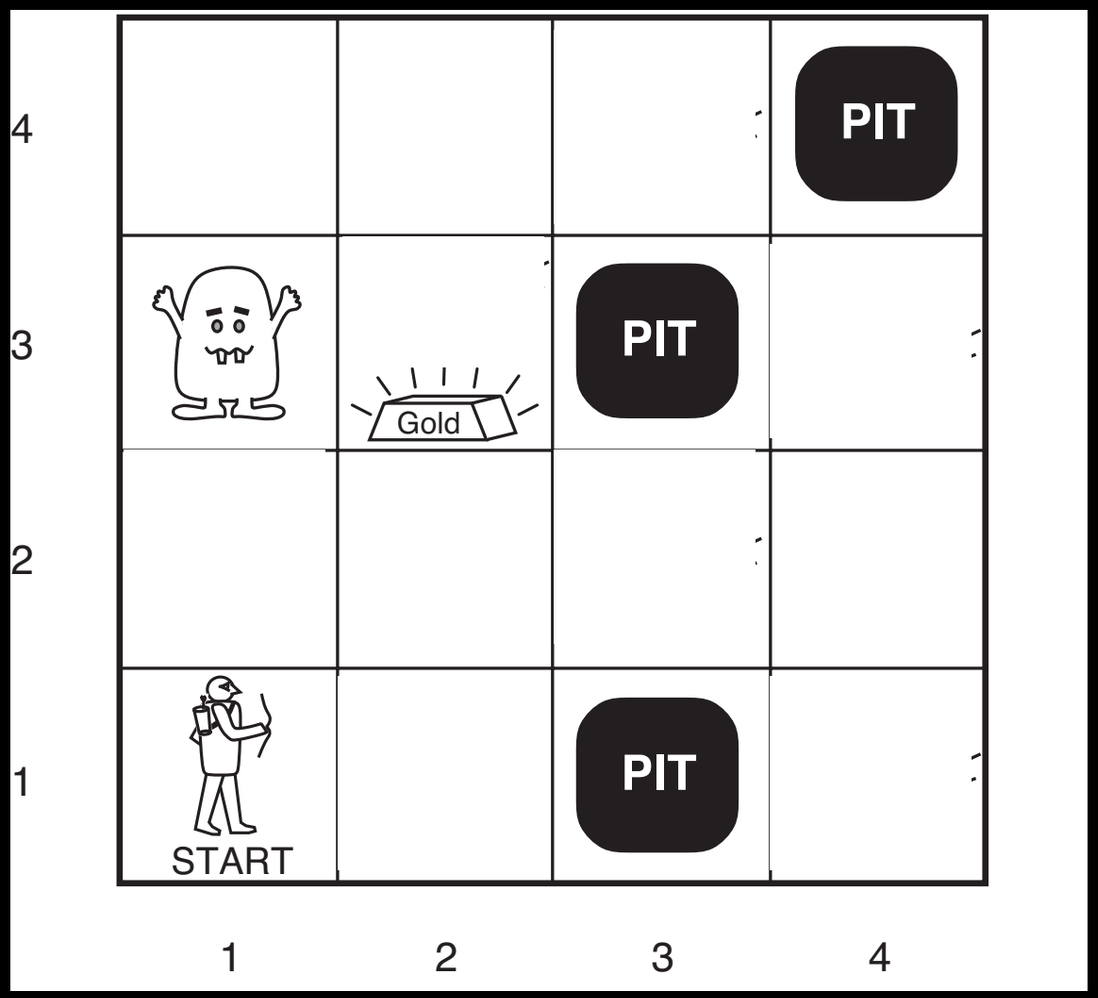

```
Boolean[][] P =
    new Boolean[n][n];
Boolean[][] G =
    new Boolean[n][n];
Boolean[][] W =
    new Boolean[n][n];


// P[i][j] == Boolean.TRUE
//    if there's a pit at [i,j]
// P[i][j] == Boolean.FALSE
//    if there's no pit at [i,j]
// P[i][j] == null
//    if we don't know
// G[i][j] ditto for gold
// W[i][j] ditto for wumpus
```

Representation



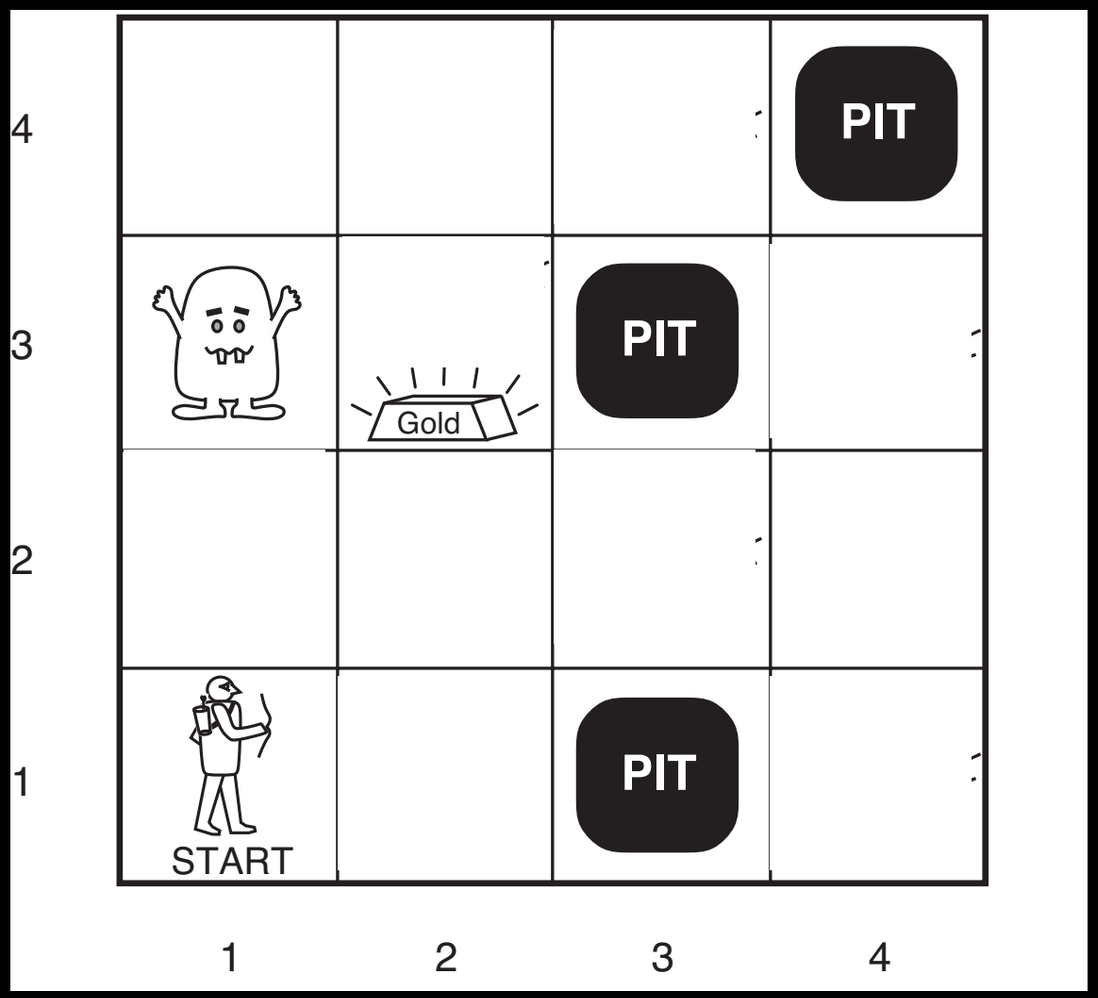

```
Position pos = <i, j>
```

Representation



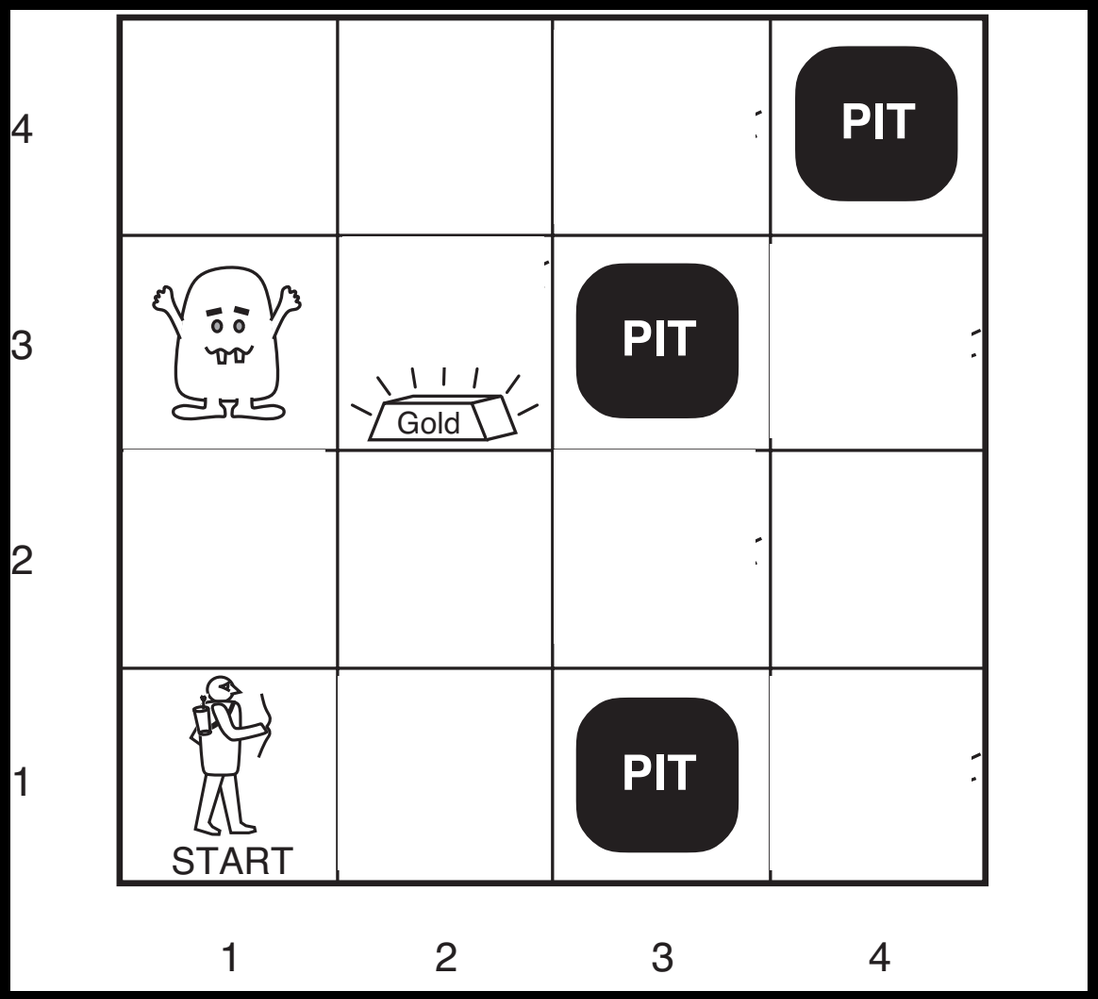

```
Boolean[][] At =
     new Boolean[n][n];

// At[i,j] == Boolean.TRUE
//    if agent is at location [i,j]
//    else Boolean.FALSE
```

Representation



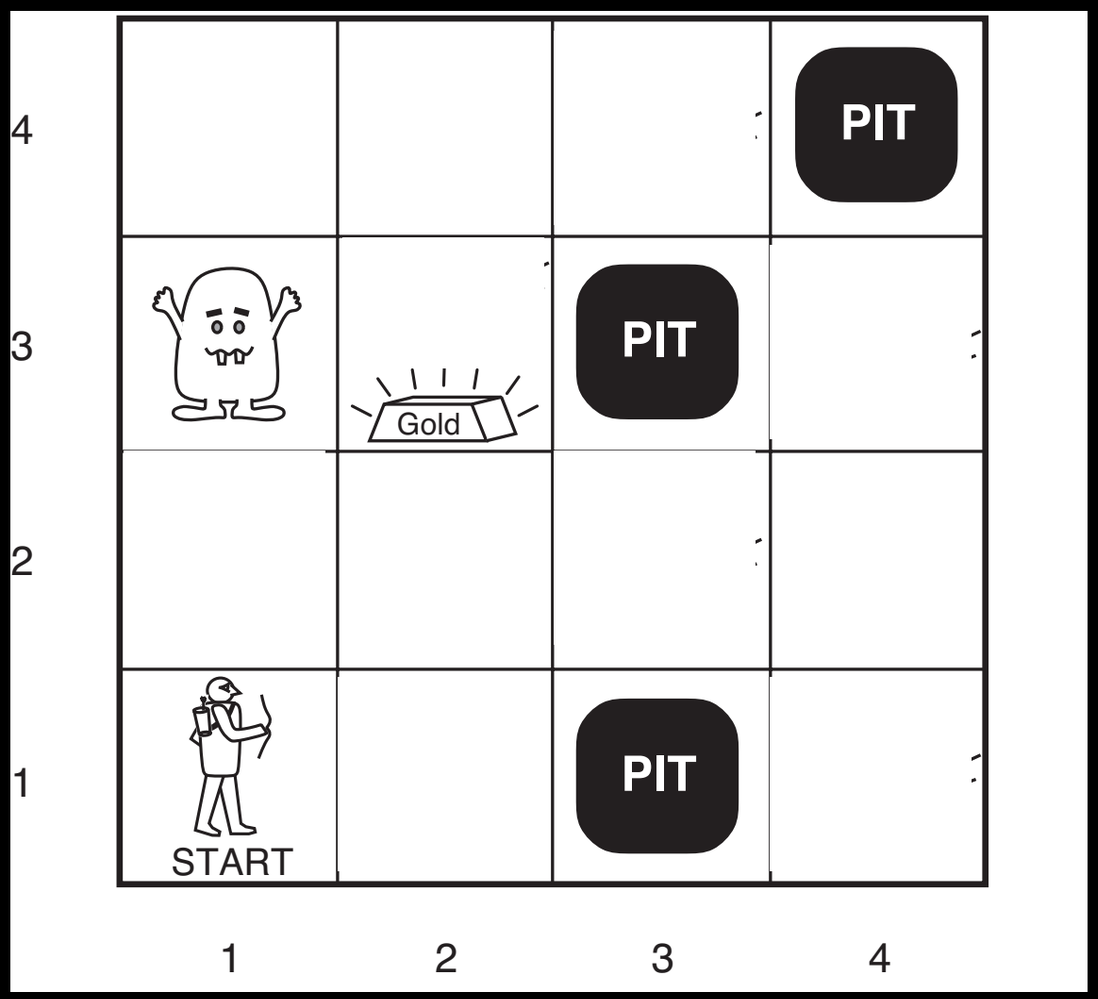

```
enum Orientation = { N,S,E,W };

Orientation orientation;
```

Representation



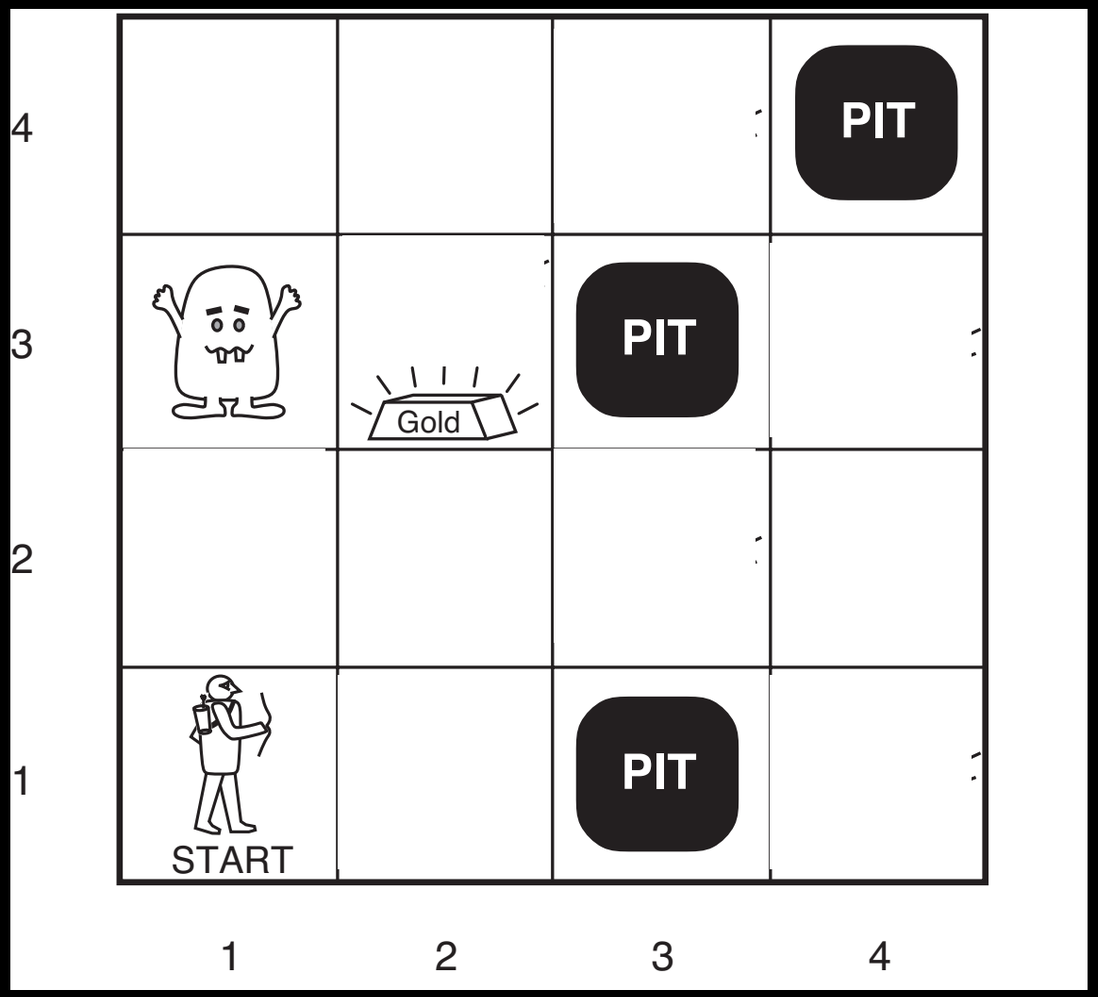

```
Boolean FacingN, FacingS,
    FacingE, FacingW;

FacingN == Boolean.TRUE
    if agent is facing north
...
```

Representation



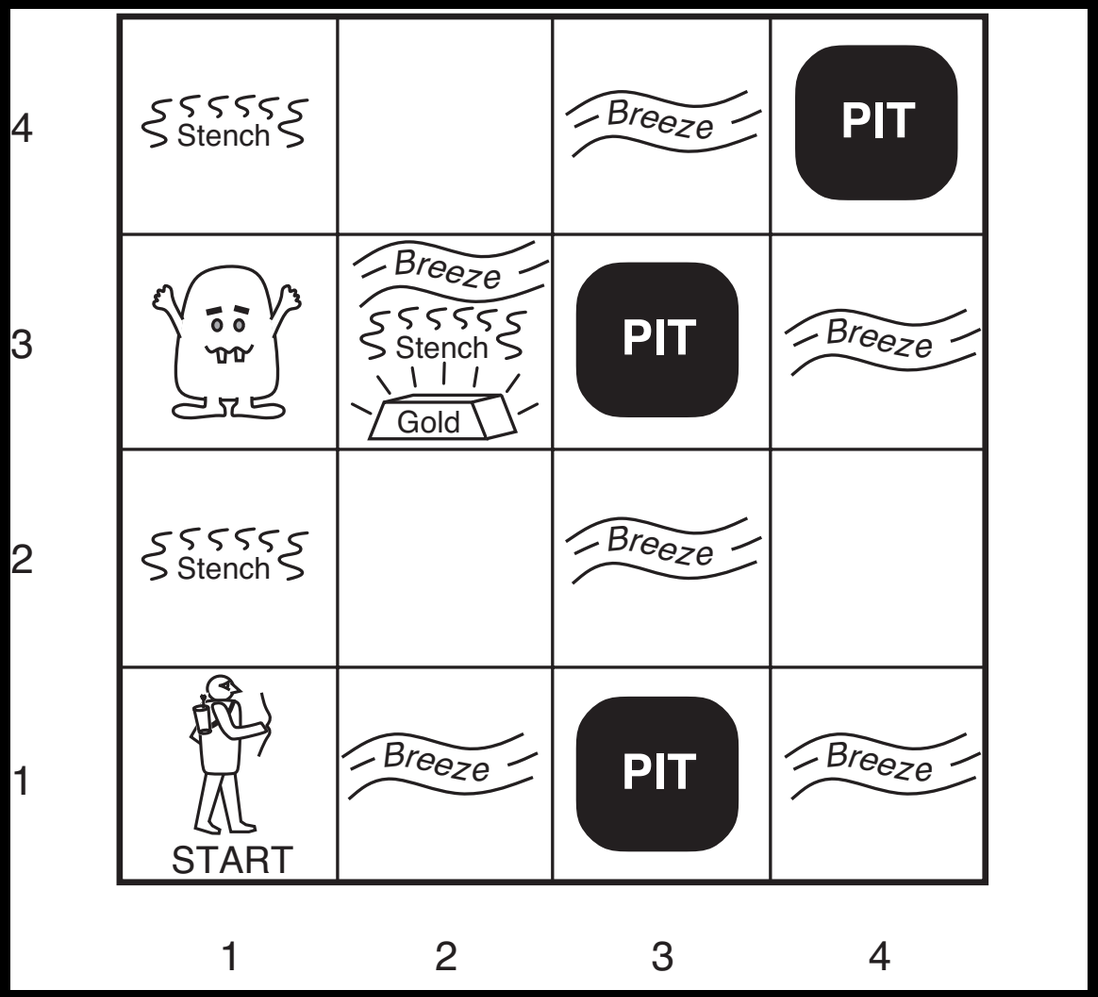

```
Boolean[][] P =
    new Boolean[n][n];
Boolean[][] G =
    new Boolean[n][n];
Boolean[][] W =
    new Boolean[n][n];
Boolean[][] At =
    new Boolean[n][n];
Boolean FacingN, FacingS,
    FacingE, FacingW;
```

Hunt The Wumpus



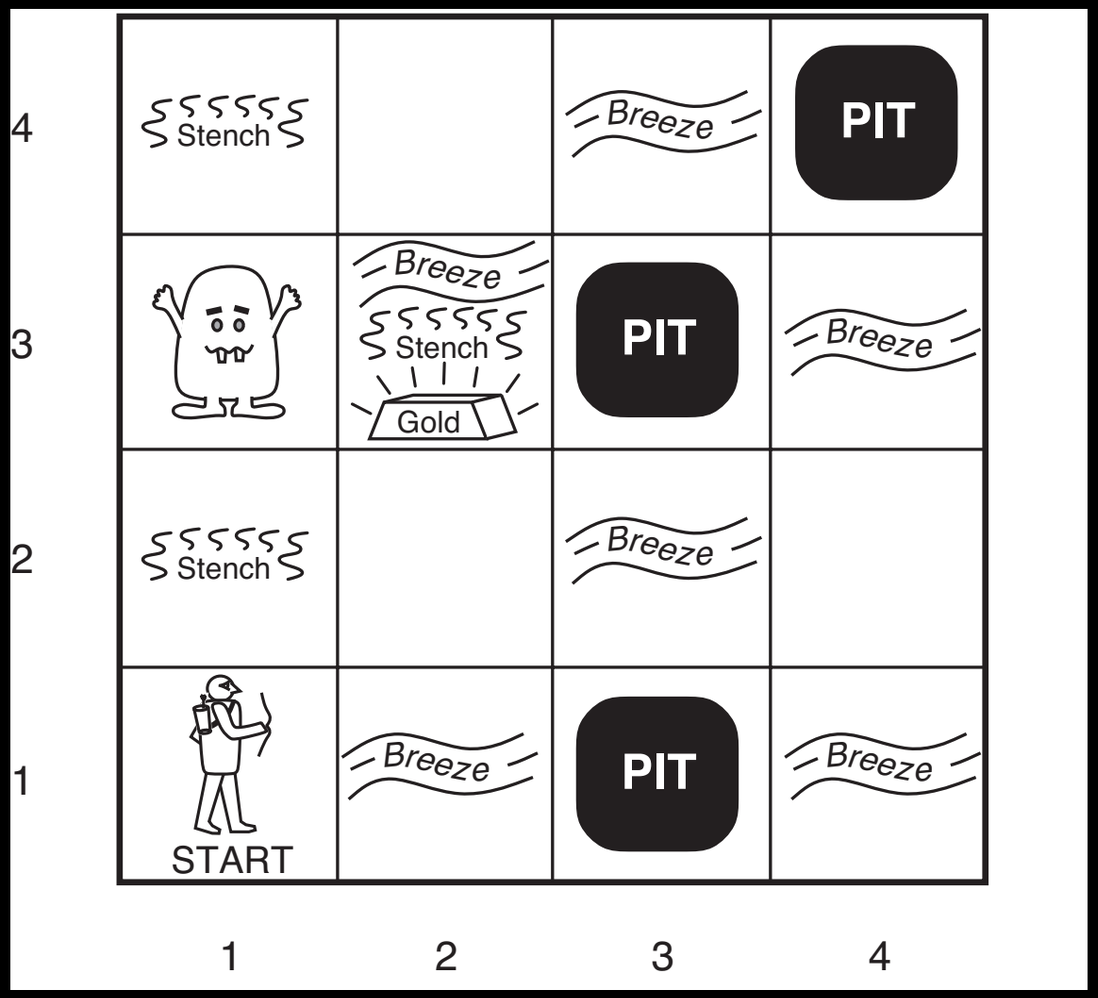
- Sensors (percepts):
  - Stench - in or adjacent to Wumpus
  - Breeze - adjacent to pit
  - Glitter - in gold square
  - Bump - hit wall
  - Scream - Wumpus killed

Representation



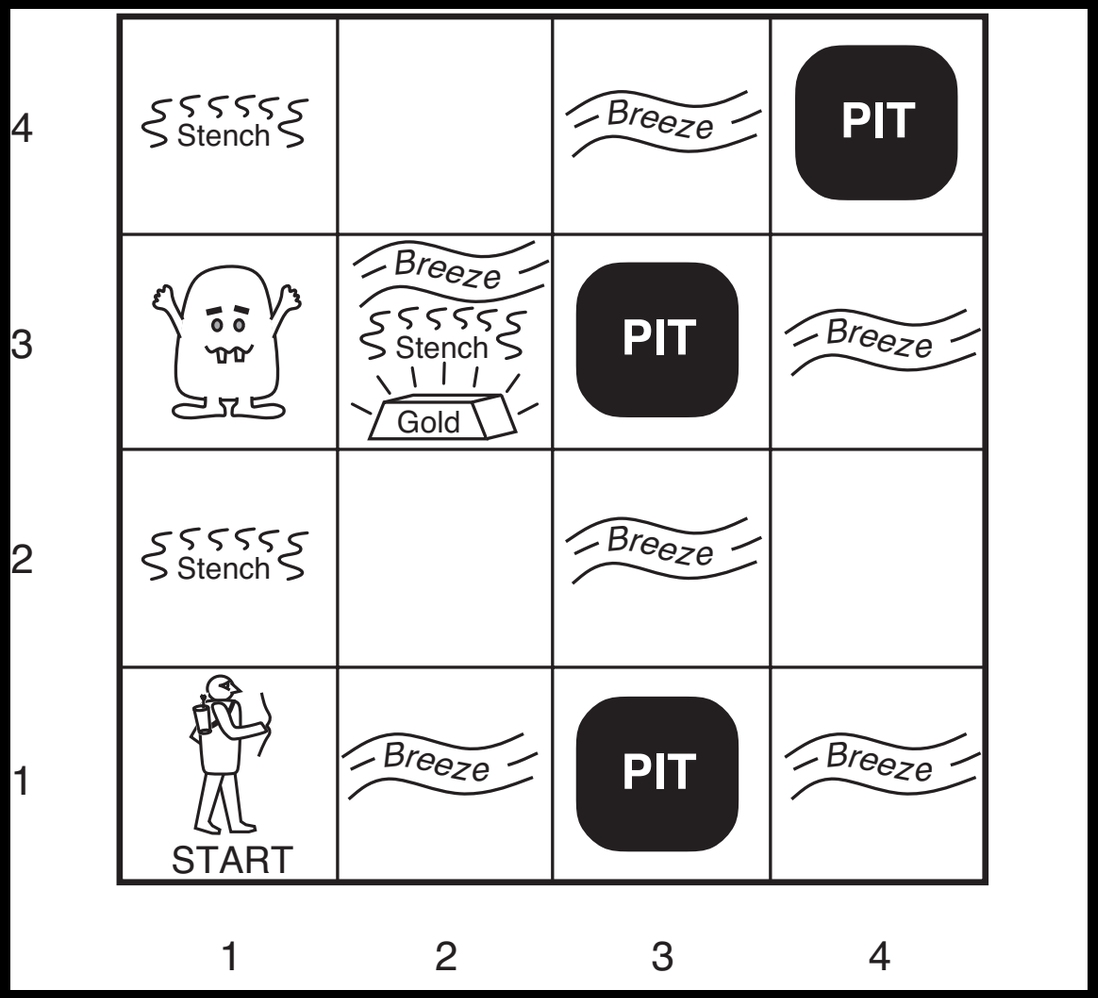

```
boolean[][][] sense =
    new boolean[n][n][5];

final int STENCH = 0;
final int BREEZE = 1;
final int GLITTER = 2;
final int BUMP = 3;
final int SCREAM = 4;
```

Representation



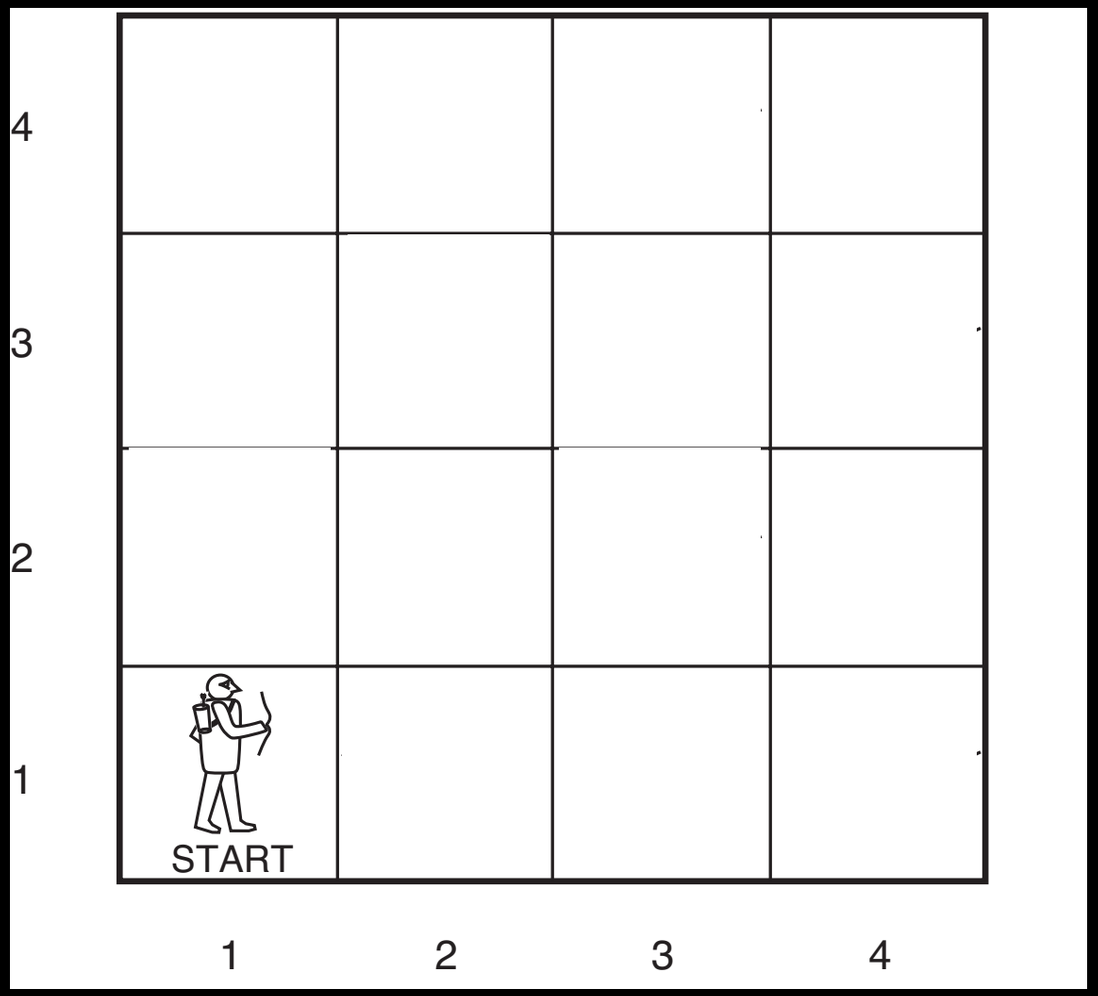

```
Boolean[][] S =
    new Boolean[n][n];
Boolean[][] B =
    new Boolean[n][n];
...

// S[i][j] == true
//    if the agent perceived a
//                   stench at [i,j]
// B[i][j] == true
//    if the agent perceived a
//                   breeze at [i,j]
...
```

Representation




```
Boolean[][] P =
    new Boolean[n][n];
Boolean[][] G =
    new Boolean[n][n];
Boolean[][] W =
    new Boolean[n][n];
Boolean[][] At =
    new Boolean[n][n];
Boolean FacingN, FacingS,
    FacingE, FacingW;
Boolean[][] S =
    new Boolean[n][n];
Boolean[][] B =
    new Boolean[n][n];
```

Hunt The Wumpus



1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
1,1 **A** **OK**	2,1	3,1	4,1

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

```
At[1][1] = true;  P[1][1] = false;
At[1][2] = false; W[1][1] = false;
...
```

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
1,1 **A** **OK**	2,1	3,1	4,1

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

```
P[1][1] = false;
W[1][1] = false;
B[1][1] = false;
S[1][1] = false;
```

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 **OK**	2,2	3,2	4,2
1,1 **A** **OK**	2,1 **OK**	3,1	4,1

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

```
P[1][1] = false;
W[1][1] = false;
B[1][1] = false;
S[1][1] = false;
W[1][2] = false;
P[1][2] = false;
W[2][1] = false;
P[2][1] = false;
```

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 **OK**	2,2	3,2	4,2
1,1 **V** **OK**	2,1 **A** **OK**	3,1	4,1

```
P[1][1] = false;
W[1][1] = false;
B[1][1] = false;
S[1][1] = false;
W[1][2] = false;
P[1][2] = false;
W[2][1] = false;
P[2][1] = false;
```

```
B[2][1] = true;
S[2][1] = false;
```

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

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

```
P[1][1] = false;     B[2][1] = true;
W[1][1] = false;     S[2][1] = false;
B[1][1] = false;
S[1][1] = false;
W[1][2] = false;
P[1][2] = false;
W[2][1] = false;
P[2][1] = false;
```

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

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

```
P[1][1] = false;
W[1][1] = false;
B[1][1] = false;
S[1][1] = false;
W[1][2] = false;
P[1][2] = false;
W[2][1] = false;
P[2][1] = false;

B[2][1] = true;
S[2][1] = false;
B[1][2] = false;
S[1][2] = true;
W[1][3] = true;
P[2][2] = false;
P[3][1] = true;
```

1,4	2,4	3,4	4,4
1,3 **W!**	2,3	3,3	4,3
1,2 **A** **S** **OK**	2,2 **OK**	3,2	4,2
1,1 **V** **OK**	2,1 **B** **V** **OK**	3,1 **P!**	4,1

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

```
P[1][1] = false;
W[1][1] = false;
B[1][1] = false;
S[1][1] = false;
W[1][2] = false;
P[1][2] = false;
W[2][1] = false;
P[2][1] = false;

B[2][1] = true;
S[2][1] = false;
B[1][2] = false;
S[1][2] = true;
W[1][3] = true;
P[2][2] = false;
P[3][1] = true;
```

Background Knowledge

- General properties (or rules) of the world
- "In partially observable environments, combine general knowledge with current percepts to infer hidden aspects of the current state prior to selecting actions."

If you perceive a stench, then there the wumpus is in an adjacent square.

If the wumpus is in an adjacent square, then you will perceive a stench.

If you perceive a stench, then there the wumpus is in an adjacent square.

```
if At[i,j] and S[i,j]
then (W[i-1,j] or W[i+1,j] or W[i,j-1] or W[i,j+1])
```

If the wumpus is in an adjacent square, then you will perceive a stench.

```
if At[i,j] and (W[i-1,j] or W[i+1,j] or W[i,j-1] or W[i,j+1])
then S[i,j]
```

Warning: not a real programming language

Boolean CSP

- Variables describing attributes or features of the world
  - Factored representation
  - Domains: Boolean → { true, false}
  - Constraints: Identify possible combinations of the boolean variables

If you perceive a stench, then there the wumpus is in an adjacent square.

```
if (At[i,j] and S[i,j])
then (W[i-1,j] or W[i+1,j] or W[i,j-1] or W[i,j+1])
```

If the wumpus is in an adjacent square, then you will perceive a stench.

```
if (At[i,j] and (W[i-1,j] or W[i+1,j] or W[i,j-1] or W[i,j+1]))
then S[i,j]
```

Warning: not a real programming language

A Programming Language for Knowledge

A Programming Language for Knowledge

- Syntax:
  - What counts as a well-formed statement, formula, sentence, or program
- Semantics:
  - What these statements, formulas, sentences, or programs mean

Syntax

- X is true
- X is not true
- Both X and Y are true
- Either X or Y are true
- If X is true then Y is true
- X is true if and only if Y is true

Syntax

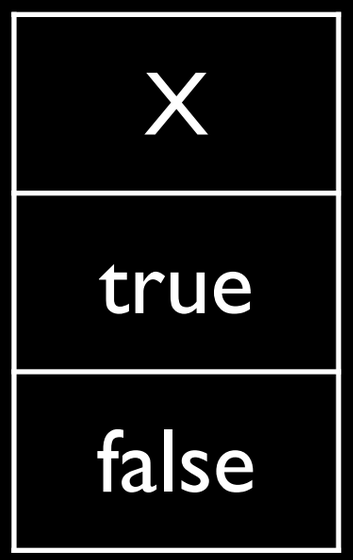
- X
- !X
- X && Y
- X || Y
- If X then Y
- X iff Y
- Parentheses: “(” and “)”

If you perceive a stench, then there the wumpus is in an adjacent square.

```
if (At[i,j] && S[i,j])
then (W[i-1,j] || W[i+1,j] || W[i,j-1] || W[i,j+1])
```

If the wumpus is in an adjacent square, then you will perceive a stench.

```
if (At[i,j] && (W[i-1,j] || W[i+1,j] || W[i,j-1] || W[i,j+1]))
then S[i,j]
```


X
true
false

X	!X
true	false
false	true

X	Y	!X
true		false
false		true

X	Y	!X
true	true	false
false	true	true
true	false	
false	false	

X	Y	!X	X && Y
true	true	false	
false	true	true	
true	false		
false	false		

X	Y	!X	X && Y
true	true	false	true
false	true	true	false
true	false		false
false	false		false

X	Y	!X	X && Y	X || Y
true	true	false	true	
false	true	true	false	
true	false		false	
false	false		false	

X	Y	!X	X && Y	X || Y
true	true	false	true	
false	true	true	false	true
true	false		false	true
false	false		false	false

X	Y	!X	X && Y	X || Y
true	true	false	true	true
false	true	true	false	true
true	false		false	true
false	false		false	false

X	Y	!X	X && Y	X || Y	ifXthenY
true	true	false	true	true	
false	true	true	false	true	
true	false		false	true	
false	false		false	false	

X	Y	!X	X && Y	X || Y	ifXthenY
true	true	false	true	true	true
false	true	true	false	true	
true	false		false	true	
false	false		false	false	

X	Y	!X	X && Y	X || Y	ifXthenY
true	true	false	true	true	true
false	true	true	false	true	
true	false		false	true	false
false	false		false	false	

X	Y	!X	X && Y	X || Y	ifXthenY
true	true	false	true	true	true
false	true	true	false	true	true
true	false		false	true	false
false	false		false	false	true

X	Y	!X	X && Y	X || Y	ifXthenY	X iff Y
true	true	false	true	true	true	
false	true	true	false	true	true	
true	false		false	true	false	
false	false		false	false	true	

X	Y	!X	X && Y	X || Y	ifXthenY	X iff Y
true	true	false	true	true	true	true
false	true	true	false	true	true	
true	false		false	true	false	
false	false		false	false	true	

X	Y	!X	X && Y	X || Y	ifXthenY	X iff Y
true	true	false	true	true	true	true
false	true	true	false	true	true	
true	false		false	true	false	
false	false		false	false	true	true

X	Y	!X	X && Y	X || Y	ifXthenY	X iff Y
true	true	false	true	true	true	true
false	true	true	false	true	true	false
true	false		false	true	false	
false	false		false	false	true	true

X	Y	!X	X && Y	X || Y	ifXthenY	X iff Y
true	true	false	true	true	true	true
false	true	true	false	true	true	false
true	false		false	true	false	false
false	false		false	false	true	true

If you perceive a stench, then there the wumpus is in an adjacent square.

```
if (At[i,j] && S[i,j])
then (W[i-1,j] || W[i+1,j] || W[i,j-1] || W[i,j+1])
```

If the wumpus is in an adjacent square, then you will perceive a stench.

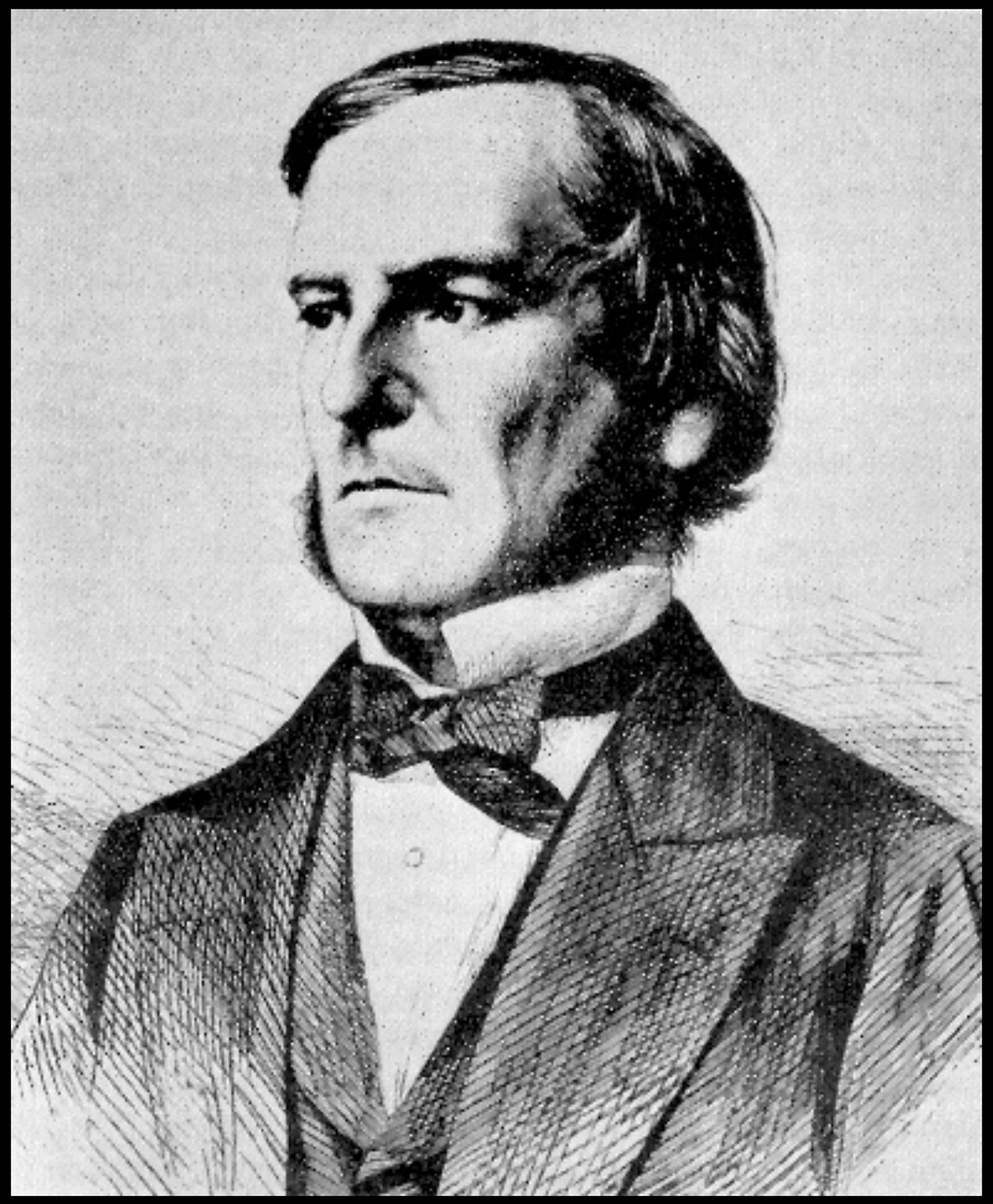
```
if (At[i,j] && (W[i-1,j] || W[i+1,j] || W[i,j-1] || W[i,j+1]))
then S[i,j]
```

Semantics

- X
- !X
- X && Y
- X || Y
- If X then Y
- X iff Y
- Parentheses: “(“ and “)”

X	Y	!X	X && Y	X || Y	if X then	X iff Y
true	true	false	true	true	true	true
false	true	true	false	true	true	false
true	false		false	true	false	false
false	false		false	false	true	true

Aristole
(384BC - 332BC)



George Boole
(1815-1864)

Propositional Logic

Proposition

- Something that can be true or false
  - “it is raining”
  - “socrates is a man”
  - “there is a wumpus at location 1,3 and I perceived a stench at location 1,2”
  - “either there is a pit at 2,2 or there is a wumpus at 2,2 and there is a pit at 2,3”

Atomic Proposition

- “it is raining”
- “socrates is a man”
- “there is a wumpus at location 1,3”
- P, Q, R, ...

Boolean variables!

Connectives

- Connect propositions into larger propositions that can also be true or false
  - "there is a wumpus at location 1,3 and I perceived a stench at location 1,2"
  - "either there is a pit at 2,2 or there is a wumpus at 2,2 and there is a pit at 2,3"

Connectives

X	P
!X	¬P
X && Y	P ∧ Q
X || Y	P ∨ Q
If X then Y	P ⇒ Q
X iff Y	P ⇔ Q

Parentheses: “(“ and “)”

P	Q	¬P	P∧Q	P∨Q	P⇒Q	P⇔Q
true	true	false	true	true	true	true
false	true	true	false	true	true	false
true	false		false	true	false	false
false	false		false	false	true	true

You perceive a breeze at [1,1] if and only if there is a pit at [2,1] or there is a pit at [2,2]

You perceive a breeze at [1,1] if and only if there is a pit at [2,1] or there is a pit at [1,2]

P: You perceive a breeze at [1,1]
Q: There is a pit at [1,2]
R: There is a pit at [2,1]

You perceive a breeze at [1,1] if and only if there is a pit at [2,1] or there is a pit at [1,2]

$B_{1,1}$: You perceive a breeze at [1,1]
$P_{1,2}$: There is a pit at [1,2]
$P_{2,1}$: There is a pit at [2,1]

You perceive a breeze at [1,1] if and only if there is a pit at [2,1] or there is a pit at [1,2]

$B_{1,1}$: You perceive a breeze at [1,1]
$P_{1,2}$: There is a pit at [1,2]
$P_{2,1}$: There is a pit at [2,1]

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

Truth Table

B	P	P	P	B
true	true	true	true	true
false	true	true	true	false
true	false	true	true	true
false	false	true	true	false
true	true	false	true	true
false	true	false	true	false
true	false	false	false	false
false	false	false	false	true

Propositional Logic (So Far)

- Syntax: how to write out legal statements in the language
- Semantics: whether a statement is true or false given the truth/falsity of the atomic propositions
- Application: Representing knowledge of the wumpus world

THE OFFICIAL

METALLICA

ILLUSTRATED CHRONICLE

SO WHAT!

THE GOOD, THE MAD AND THE UGLY

EDITED BY STEFFAN CHIRAZI

Inference

- Compute the consequences of your knowledge
- Make implicit knowledge explicit
- Given what you've represented explicitly, compute what else is true (or false)

If you perceive a stench, then there the wumpus is in an adjacent square.

```
if (At[i,j] and S[i,j])
then (W[i-1,j] or W[i+1,j] or W[i,j-1] or W[i,j+1])
```

$$(At_{i,j} \wedge S_{i,j}) \Rightarrow (W_{i-1,j} \vee W_{i+1,j} \vee W_{i,j-1} \vee W_{i,j+1})$$

If the wumpus is in an adjacent square, then you will perceive a stench.

```
if (At[i,j] and (W[i-1,j] or W[i+1,j] or W[i,j-1] or W[i,j+1]))
then S[i,j]
```

$$(At_{i,j} \wedge (W_{i-1,j} \vee W_{i+1,j} \vee W_{i,j-1} \vee W_{i,j+1})) \Rightarrow S_{i,j}$$

$$(At_{1,1} \wedge S_{1,1}) \Rightarrow (W_{2,1} \vee W_{1,2})$$
$$(At_{1,2} \wedge S_{1,2}) \Rightarrow (W_{1,1} \vee W_{2,2} \vee W_{1,3})$$
$$(At_{2,1} \wedge S_{2,1}) \Rightarrow (W_{1,1} \vee W_{2,2} \vee W_{3,1})$$
$$(At_{2,2} \wedge S_{2,2}) \Rightarrow (W_{1,2} \vee W_{2,1} \vee W_{2,3} \vee W_{3,2})$$
$$\ldots$$

$$(At_{1,1} \wedge (W_{2,1} \vee W_{1,2})) \Rightarrow S_{1,1}$$
$$(At_{1,2} \wedge (W_{2,1} \vee W_{2,2} \vee W_{1,3})) \Rightarrow S_{1,2}$$
$$(At_{2,1} \wedge (W_{1,1} \vee W_{2,2} \vee W_{3,1})) \Rightarrow S_{2,1}$$
$$(At_{2,2} \wedge (W_{1,2} \vee W_{2,1} \vee W_{2,3} \vee W_{3,2})) \Rightarrow S_{2,2}$$
$$\ldots$$

$$At_{1,1}$$

$$\neg S_{1,1}$$

$$S_{1,1} \iff (W_{2,1} \vee W_{1,2})$$

$\neg W_{1,2}$?

$\neg W_{2,1}$?

Inference

- Given a set of statements in propositional logic (facts and background knowledge)
- Test whether some other statement is true

Boolean CSP

- Variables describing attributes or features of the world
  - Factored representation
- Domains: Boolean → { true, false}
- Constraints: Identify possible combinations of the boolean variables
- Solution: Complete assignment consistent with the constraints

Model (Possible World)

- Assignment of true or false to all the propositional variables

Model (Possible World)

- Assignment of true or false to all the propositional variables
- A model satisfies a sentence if it makes the sentence true
- "A model of the sentence"

Satisfaction

B	P	P	P	B
true	true	true	true	true
false	true	true	true	false
true	false	true	true	true
false	false	true	true	false
true	true	false	true	true
false	true	false	true	false
true	false	false	false	false
false	false	false	false	true

Inference

- Given a set of statements in propositional logic (facts and background knowledge)
- Test whether some other statement is true

Entailment

- If α entails β: $\alpha \models \beta$
  - Whenever α is true, so is β
  - Every model of α is a model of β
  - Models(α) ⊆ Models(β)
  - α is at least as strong an assertion as β
    - Rules out no fewer possible worlds

Entailment

- If α entails β: $\alpha \models \beta$
  - Whenever α is true, so is β
  - Every model of α is a model of β
  - Models(α) ⊆ Models(β)
  - α is at least as strong an assertion as β
    - Rules out no fewer possible worlds

Entailment

- If α entails β: $\alpha \models \beta$
  - Whenever α is true, so is β
  - Every model of α is a model of β
  - Models(α) ⊆ Models(β)
  - α is at least as strong an assertion as β
    - Rules out no fewer possible worlds

P	Q	...	α	β
true	true	...	true	true
false	true	...	false	false
true	false	...	true	true
false	false	...	false	false
true	true	...	false	true
false	true	...	true	true
true	false	...	false	false
false	false	...	false	true

P	Q	...	α	β
true	true	...	true	true
false	true	...	false	false
true	false	...	true	true
false	false	...	false	false
true	true	...	false	true
false	true	...	true	true
true	false	...	false	false
false	false	...	false	true

Model Checking

P	Q	...	α	β
true	true	...	true	true
false	true	...	false	false
true	false	...	true	true
false	false	...	false	false
true	true	...	false	true
false	true	...	true	true
true	false	...	false	false
false	false	...	false	true

Inference

- If α entails β
  - Then when you know α is true you also know β is true
    - Even though you didn't write it down explicitly!
- α might be your knowledge, β your question

Entailment

- If α entails β: $\alpha \models \beta$
  - Whenever α is true, so is β
  - Every model of α is a model of β
  - Models(α) ⊆ Models(β)
  - α is at least as strong an assertion as β
    - Rules out no fewer possible worlds

P	Q	...	α	β
true	true	...	true	true
false	true	...	false	false
true	false	...	true	true
false	false	...	false	false
true	true	...	false	true
false	true	...	true	true
true	false	...	false	false
false	false	...	false	true

Computing Entailment

```
boolean tt_entails(Set<Sentence> kb, Sentence alpha) {
    List<Symbol> symbols = new List(kb.getSymbols());
    symbols.append(alpha.getSymbols());
    return tt_check_all(kb, alpha, symbols, new Model());
}

boolean tt_check_all(Set<Sentence> kb, Sentence alpha,
                     List<Symbol> symbols, Model model) {
    if (symbols.isEmpty()) {
        if (model.satisfies(kb)) {
            return model.satisfies(alpha);
        } else {
            return true;
        }
    } else {
        Symbol p = symbols.removeFirst();
        return (tt_check_all(kb, alpha, symbols,
                             model.clone().assign(p, Boolean.TRUE)) &&
                tt_check_all(kb, alpha, symbols,
                             model.clone().assign(p, Boolean.FALSE)));
    }
}
```

```
class Model extends Hashtable implements Cloneable {
    public Model() {
    }
    public void assign(Symbol symbol, Boolean value) {
        return this.put(symbol, value);
    }
    public boolean satisfies(Sentence sentences) {
        ...
    }
    public boolean satisfies(Set<Sentence> sentences) {
        for (Sentence s : sentences) {
            if (!this.satisfies(s)) {
                return false;
            }
        }
        return  false;
    }
}
```

Computing Entailment

```
boolean tt_entails(Set<Sentence> kb, Sentence alpha) {
    List<Symbol> symbols = new List(kb.getSymbols());
    symbols.append(alpha.getSymbols());
    return tt_check_all(kb, alpha, symbols, new Model());
}

boolean tt_check_all(Set<Sentence> kb, Sentence alpha,
                     List<Symbol> symbols, Model model) {
    if (symbols.isEmpty()) {
        if (model.satisfies(kb)) {
            return model.satisfies(alpha);
        } else {
            return true;
        }
    } else {
        Symbol p = symbols.removeFirst();
        return (tt_check_all(kb, alpha, symbols,
                             model.clone().assign(p, Boolean.TRUE)) &&
                tt_check_all(kb, alpha, symbols,
                             model.clone().assign(p, Boolean.FALSE)));
    }
}
```

Propositional Logic

- Programming language for knowledge
- Factored representation of state
  - Propositions, connectives, sentences
- Model (possible world) = assignment of true or false to propositions
- Entailment: $\alpha \models \beta$
  - Every model of α is a model of β

For next time:
AIMA Section 7.5
INFERENCE

Next Week: Exam 1
Search