Python Programming

```
>>> text = 'UPPER PYTHON, lower python, Mixed Python'
>>> re.findall('python', text, flags=re.IGNORECASE)
['PYTHON', 'python', 'Python']
>>> re.sub('python', 'snake', text, flags=re.IGNORECASE)
'UPPER snake, lower snake, Mixed snake'
```

```
def matchcase(word):
    def replace(m):
        text = m.group()
        if text.isupper():
            return word.upper()
        elif text.islower():
            return word.lower()
        elif text[0].isupper():
            return word.capitalize()
        else:
            return word
    return replace
```

```
>>> re.sub('python', matchcase('snake'), text, flags=re.IGNORECASE)
'UPPER SNAKE, lower snake, Mixed Snake'
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> str_pat = re.compile(r'\"(.*)\"')
>>> text1 = 'Computer says "no."'
>>> str_pat.findall(text1)
['no.']
>>> text2 = 'Computer says "no." Phone says "yes."'
>>> str_pat.findall(text2)
['no." Phone says "yes.']
>>>
```

```
>>> str_pat = re.compile(r'\"(.*?)\"')
>>> str_pat.findall(text2)
['no.', 'yes.']
>>>
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> comment = re.compile(r'/\*(.*?)\*/')
>>> text1 = '/* this is a comment */'
>>> text2 = '''/* this is a
...               multiline comment */
... '''
>>>
>>> comment.findall(text1)
[' this is a comment ']
>>> comment.findall(text2)
[]
```

```
>>> comment = re.compile(r'/\*((?:.|\n)*?)\*/')
>>> comment.findall(text2)
[' this is a\n               multiline comment ']
```

```
>>> comment = re.compile(r'/\*(.*?)\*/', re.DOTALL)
>>> comment.findall(text2)
[' this is a\n               multiline comment ']
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> s1 = 'Spicy Jalape\u00f1o'
>>> s2 = 'Spicy Jalapen\u0303o'
>>> s1
'Spicy Jalapeño'
>>> s2
'Spicy Jalapeño'
>>> s1 == s2
False
>>> len(s1)
14
>>> len(s2)
15
```

```
>>> import unicodedata
>>> t1 = unicodedata.normalize('NFC', s1)
>>> t2 = unicodedata.normalize('NFC', s2)
>>> t1 == t2
True
>>> print(ascii(t1))
'Spicy Jalape\xf1o'

>>> t3 = unicodedata.normalize('NFD', s1)
>>> t4 = unicodedata.normalize('NFD', s2)
>>> t3 == t4
True
>>> print(ascii(t3))
'Spicy Jalapen\u0303o'
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> s = '\ufb01'    # A single character
>>> s
'ﬁ'
>>> unicodedata.normalize('NFD', s)
'ﬁ'

# Notice how the combined letters are broken apart here
>>> unicodedata.normalize('NFKD', s)
'fi'
>>> unicodedata.normalize('NFKC', s)
'fi'
>>>
```

```
>>> t1 = unicodedata.normalize('NFD', s1)
>>> ''.join(c for c in t1 if not unicodedata.combining(c))
'Spicy Jalapeno'
>>>
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> import re
>>> num = re.compile('\d+')
>>> # ASCII digits
>>> num.match('123')
<_sre.SRE_Match object at 0x1007d9ed0>

>>> # Arabic digits
>>> num.match('\u0661\u0662\u0663')
<_sre.SRE_Match object at 0x101234030>

>>> arabic = re.compile('[\u0600-\u06ff\u0750-\u077f\u08a0-\u08ff]+')

>>> pat = re.compile('stra\u00dfe', re.IGNORECASE)
>>> s = 'straße'
>>> pat.match(s)                # Matches
<_sre.SRE_Match object at 0x10069d370>
>>> pat.match(s.upper())        # Doesn't match
>>> s.upper()                   # Case folds
'STRASSE'
>>>
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> # Whitespace stripping
>>> s = '   hello world  \n'
>>> s.strip()
'hello world'
>>> s.lstrip()
'hello world  \n'
>>> s.rstrip()
'   hello world'
>>>

>>> # Character stripping
>>> t = '-----hello====='
>>> t.lstrip('-')
'hello====='
>>> t.strip('-=')
'hello'
>>>
```

```
>>> s = ' hello        world   \n'
>>> s = s.strip()
>>> s
'hello       world'
>>>

>>> s.replace(' ', '')
'helloworld'
>>> import re
>>> re.sub('\s+', ' ', s)
'hello world'
>>>

with open(filename) as f:
    lines = (line.strip() for line in f)
    for line in lines:
        ...
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> s = 'pýtĥöñ\fis\tawesome\r\n'
>>> s
'pýtĥöñ\x0cis\tawesome\r\n'
>>>
```

```
>>> remap = {
...     ord('\t') : ' ',
...     ord('\f') : ' ',
...     ord('\r') : None      # Deleted
... }
>>> a = s.translate(remap)
>>> a
'pýtĥöñ is awesome\n'
>>>
```

```
>>> import unicodedata
>>> import sys
>>> cmb_chrs = dict.fromkeys(c for c in range(sys.maxunicode)
...                          if unicodedata.combining(chr(c)))
...
>>> b = unicodedata.normalize('NFD', a)
>>> b
'pýtĥöñ is awesome\n'
>>> b.translate(cmb_chrs)
'python is awesome\n'
>>>
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> digitmap = { c: ord('0') + unicodedata.digit(chr(c))
...              for c in range(sys.maxunicode)
...              if unicodedata.category(chr(c)) == 'Nd' }
...
>>> len(digitmap)
460
>>> # Arabic digits
>>> x = '\u0661\u0662\u0663'
>>> x.translate(digitmap)
'123'
>>>
```

```
>>> a
'pýtĥöñ is awesome\n'
>>> b = unicodedata.normalize('NFD', a)
>>> b.encode('ascii', 'ignore').decode('ascii')
'python is awesome\n'
>>>
```

```
def clean_spaces(s):
    s = s.replace('\r', '')
    s = s.replace('\t', ' ')
    s = s.replace('\f', ' ')
    return s
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> text = 'Hello World'
>>> text.ljust(20)
'Hello World         '
>>> text.rjust(20)
'         Hello World'
>>> text.center(20)
'    Hello World     '
>>>

>>> text.rjust(20,'=')
'=========Hello World'
>>> text.center(20,'*')
'****Hello World*****'

>>> format(text, '>20')
'         Hello World'
>>> format(text, '<20')
'Hello World         '
>>> format(text, '^20')
'    Hello World     '
```

```
>>> format(text, '=>20s')
'=========Hello World'

>>> format(text, '*^20s')
'****Hello World*****'

>>> '{:>10s} {:>10s}'.format('Hello', 'World')
'     Hello      World'
>>>

>>> x = 1.2345
>>> format(x, '>10')
'    1.2345'
>>> format(x, '^10.2f')
'   1.23   '

>>> '%-20s' % text
'Hello World         '
>>> '%20s' % text
'         Hello World'
>>>
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Artificial Intelligence

Logical Agents

In which we design agents that can form representations of the world, use a process of inference to derive new representations about the world, and use these new representations to deduce what to do.

Knowledge-Based Logical Agents

- Two central AI concepts
  - Representation of <u>knowledge</u>
  - <u>Reasoning</u> processes acting on knowledge
- Play crucial role in "Partially Observable" environments
  - Combine general knowledge with current percepts to infer hidden aspects before acting
- Aids in agent flexibility
  - Learn new knowledge for <u>new tasks</u>
  - <u>Adapt to changes</u> in environment by updating relevant knowledge

Logic

- For <u>logical</u> agents, knowledge is <u>definite</u>
  - Each proposition is either "True" or "False"
- Logic has advantage of being simple representation for knowledge-based agents
  - But limited in its ability to handle uncertainty
- We will examine propositional logic and first-order logic

Knowledge Base

- Central component is its knowledge base (KB)
  - Contains set of “sentences” or factual statements
    - Some assertions about the world expressed with a knowledge representation language
  - KB initially contains some background knowledge
    - Innate knowledge
- How to add new information to KB?
  - TELL function
  - Inference: deriving new sentences from old ones
- How to query what is known?
  - ASK function
  - Answers should follow what has been told to the KB previously

A Simple Knowledge-Based Agent

- Agent needs to know
  - Current state of world
  - How to infer unseen properties of world from percepts
  - How world evolves over time
  - What it wants to achieve
  - What its own actions do in various circumstances

“Wumpus World” Environment

- Simple environment to motivate logical reasoning
- Agent explores cave with rooms connected by passageways
- “Wumpus” beast lurking somewhere in cave
  - Eats anyone who enters its room
  - Agent has one arrow (can kill Wumpus)
- Some rooms contain bottomless pits
- Occasional heap of gold present
- Agent task
  - Enter cave, find the gold, return to entrance, and exit

Wumpus World PEAS Description

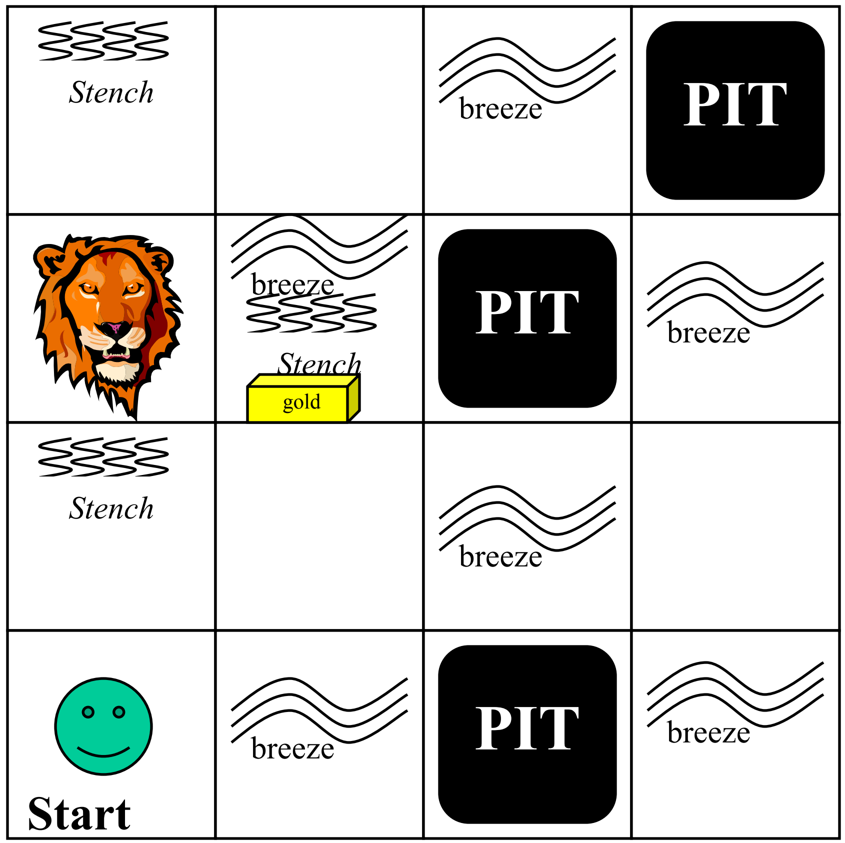
- (P)erformance measure
  - Receive +1000 for picking up gold
  - Cost of –1000 for falling into pit or being eaten by Wumpus (GAME OVER!)
  - Cost of –1 for each action taken
  - Cost of –10 for using up the only arrow
- (E)nvironment
  - 4x4 grid of rooms
  - Agent starts in square [1,1]
  - Wumpus and gold locations chosen randomly
  - Probability of square being a pit is .2
    - [0=*no*, …, 0.5=*maybe*, …, 1=*yes*]

Wumpus World PEAS Description

- (A)ctuators
  - Move forward, turn left, turn right
    - Note: die if enter pit or live wumpus square
  - Grab (gold)
  - Shoot (arrow)
    - Kills wumpus if facing its square
- (S)ensors
  - Nose: squares adjacent to wumpus are “smelly”
  - Skin/hair: Squares adjacent to pit are “breezy”
  - Eye: “Glittery” if and only if gold is in the same square
  - Percepts: [Stench, Breeze, Glitter]

Wumpus World

Lion = wumpus →



Wumpus World Characterization

- Is the world deterministic?
  - Yes, outcomes exactly specified
- Is the world fully observable?
  - No, only <u>local</u> percepts
- Is the world static?
  - Yes, Wumpus and pits do not move (though would be interesting!)
- Is the world discrete?
  - Yes, blocks/cells

Exploring a Wumpus World

A = agent

B = breeze

G = glitter, gold

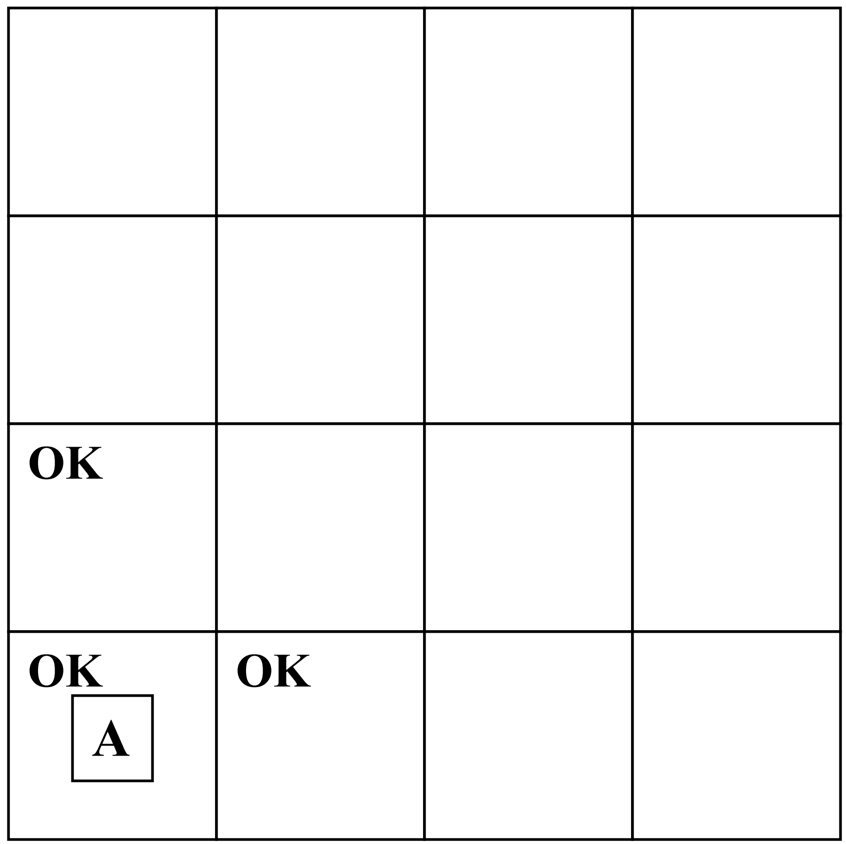
OK = safe square

P = pit

S = stench

V = visited

W = Wumpus



From local percepts, determines that {(1,1), (1,2), (2,1)} are free from danger.

Exploring a Wumpus World

A = agent

B = breeze

G = glitter, gold

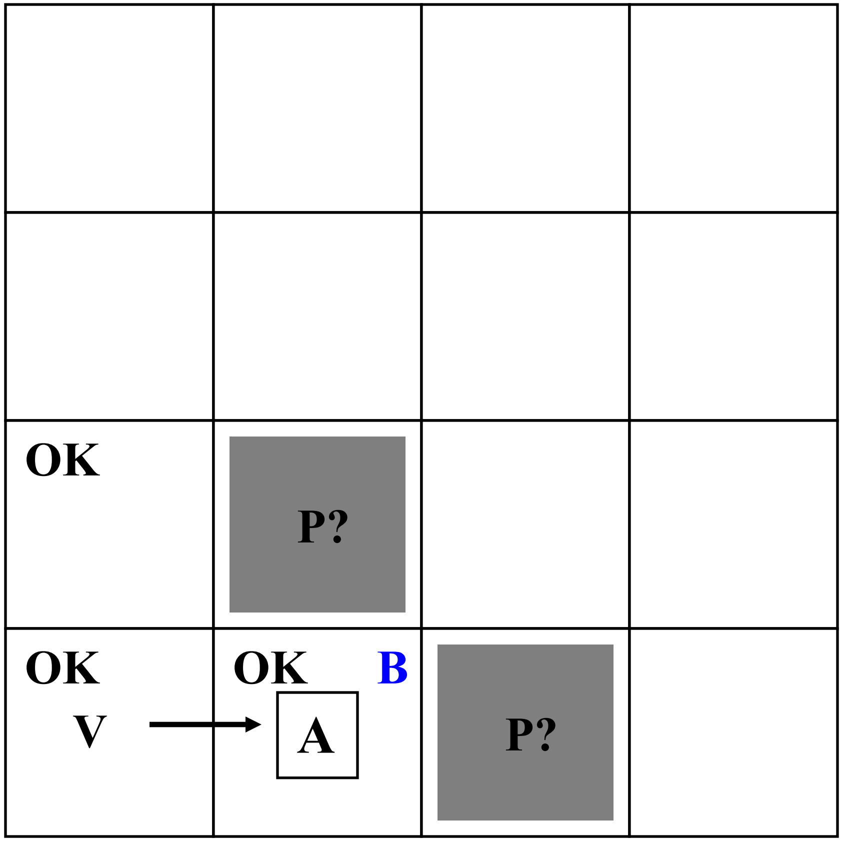
OK = safe square

P = pit

S = stench

V = visited

W = Wumpus



From breeze percept, determines that (2,2) or (3,1) is a pit. Go back to (1,1) and move up to (1,2).

Exploring a Wumpus World

A = agent

B = breeze

G = glitter, gold

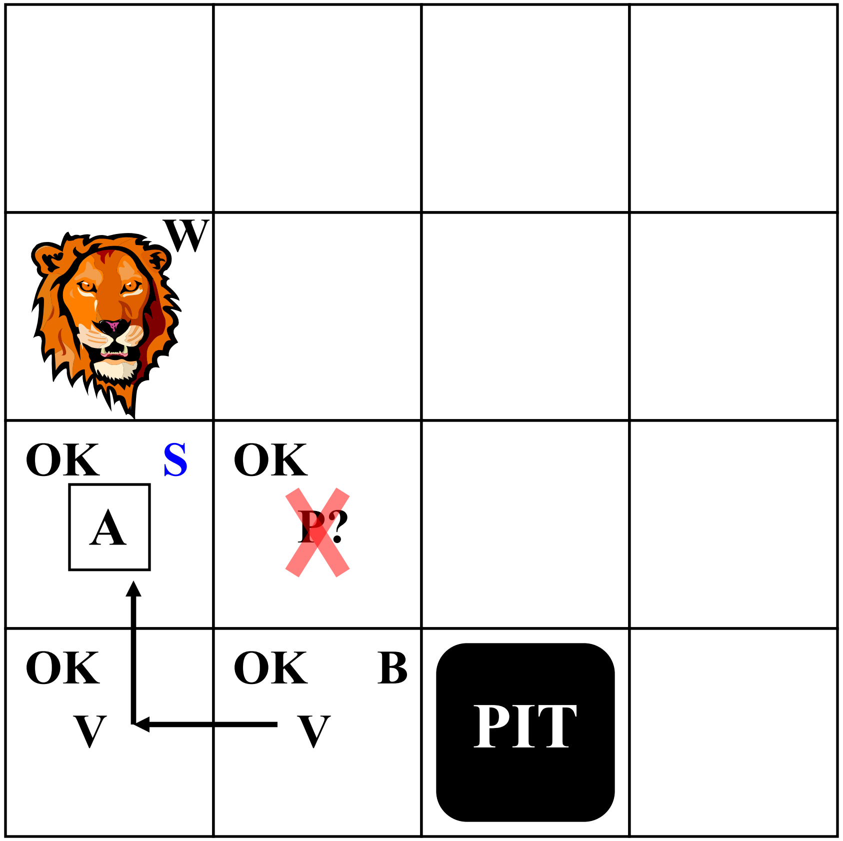
OK = safe square

P = pit

S = stench

V = visited

W = Wumpus



From stench and no-breeze percept in (1,2), determines that Wumpus in (1,3), pit in (3,1), and (2,2) clear.

Exploring a Wumpus World

A = agent

B = breeze

G = glitter, gold

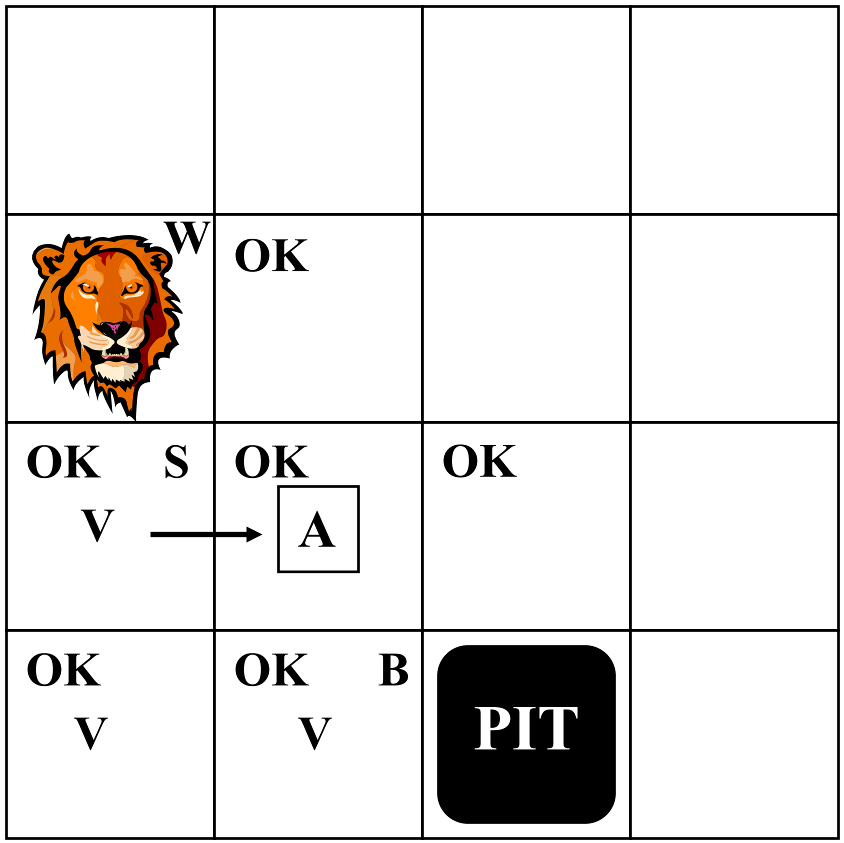
OK = safe square

P = pit

S = stench

V = visited

W = Wumpus



From local percepts, it is OK to move up or right.

Exploring a Wumpus World

A = agent

B = breeze

G = glitter, gold

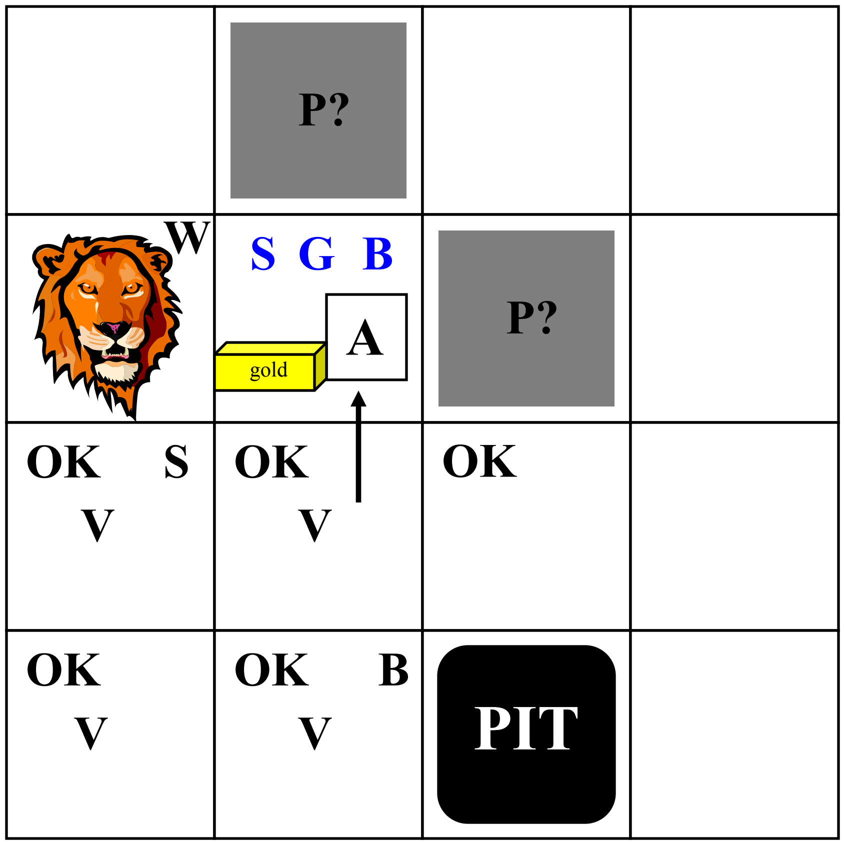
OK = safe square

P = pit

S = stench

V = visited

W = Wumpus



Found gold! No need to explore further. Time to head back.

Exploring a Wumpus World

A = agent

B = breeze

G = glitter, gold

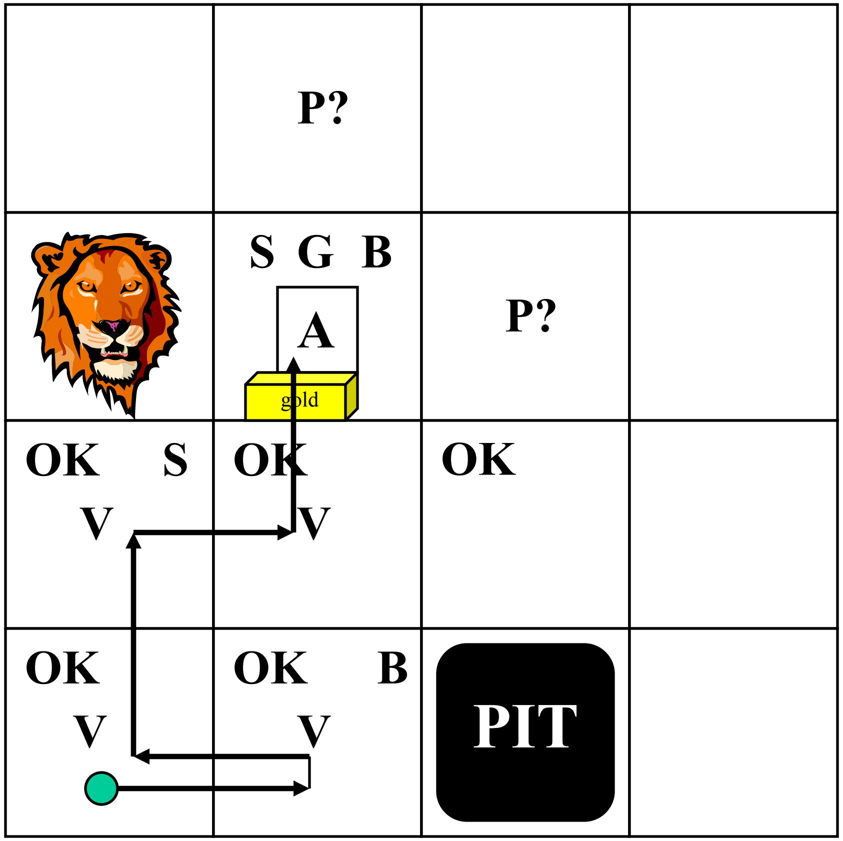
OK = safe square

P = pit

S = stench

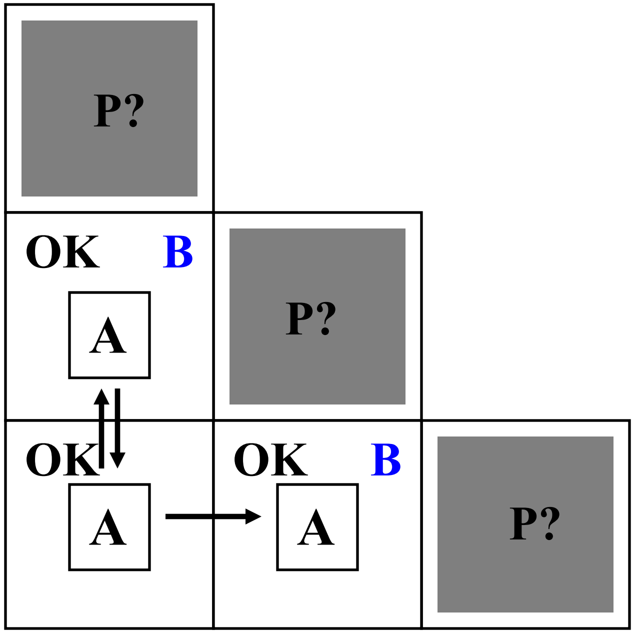
V = visited

W = Wumpus



Then go home using **OK** squares (retrace route).

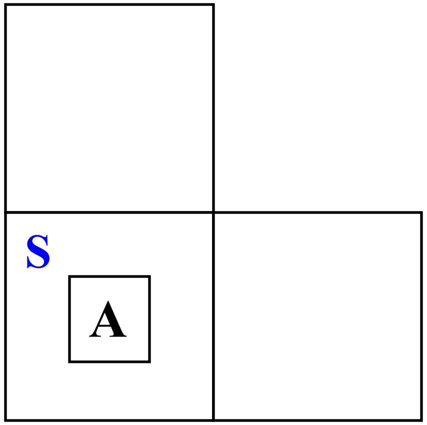
Tight Spots



Breeze in (1,2) and (2,1) → no safe actions!

Pit may actually only be in (2,2), but can't tell.

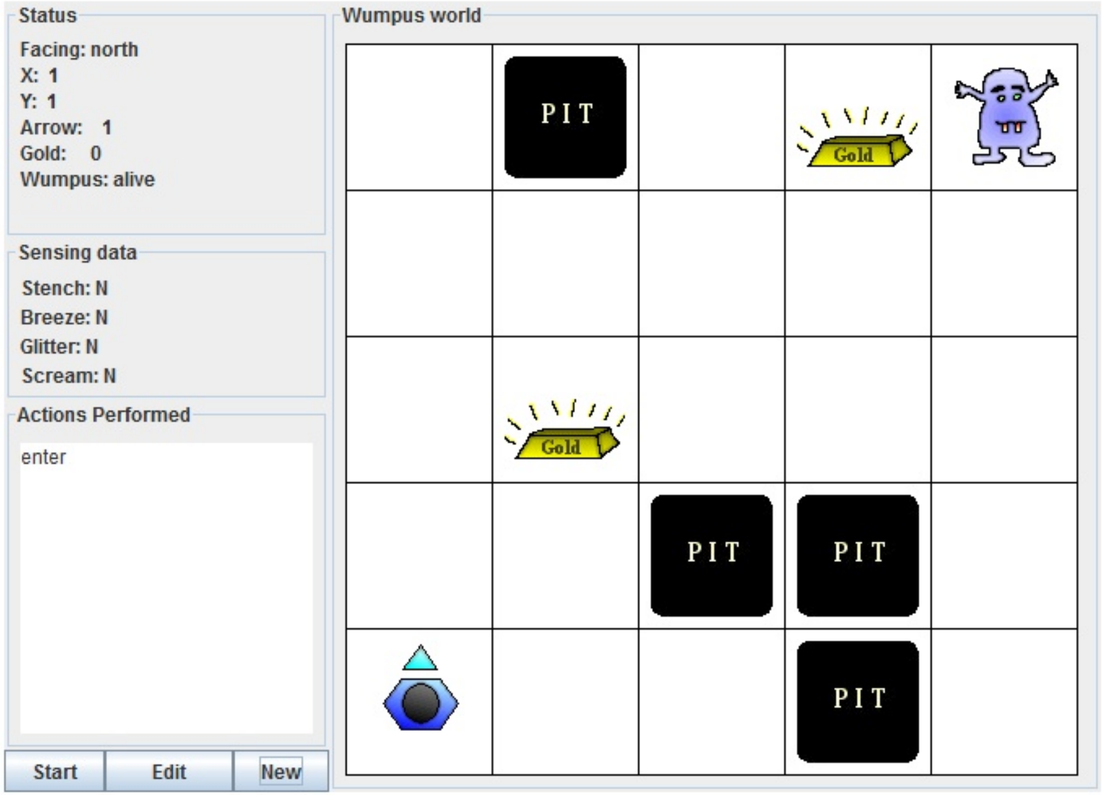
More Tight Spots



Smell in (1,1) → Cannot move!

Possible action: shoot arrow straight ahead

Online Examples



http://www.kr.tuwien.ac.at/students/prak_wumpusjava/simulator/Welcome.html

Logical Agent

- Need agent to represent beliefs
  - "There is a pit in (2, 2) or (3, 1)"
  - "There is no Wumpus in (2, 2)"
- Need to make inferences
  - If available information is correct, draw a conclusion that is guaranteed to be correct
- Need <u>representation</u> and <u>reasoning</u>
  - Support the operation of knowledge-based agent

Knowledge Representation

- For expressing knowledge in computer-tractable form
- Knowledge representation language defined by
  - **Syntax**
    - Defines the possible well-formed configurations of sentences in the language
  - **Semantics**
    - Defines the "meaning" of sentences (need interpreter)
    - Defines the truth of a sentence in a world (or model)

Provided the syntax and semantics are defined precisely, the language is called a <u>logic</u>

The Language of Arithmetic

Syntax: "$x + 2 \geq y$" is a sentence

"$x2 + y >$" is not a sentence

Semantics: $x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y

$x + 2 \geq y$ is True in a world where x=7, y=1

$x + 2 \geq y$ is False in a world where x=0, y=6

Entailment

- Want to generate new sentences that are necessarily true, given that old sentences are true
- Entailment has one fact following logically from another
- KB $\models \alpha$
  - Knowledge base (KB) "entails" sentence α
    - If α is true in all worlds where KB is true
  - The truth of α is contained in the truth of the KB
- The KB containing "*The Giants won*" and "*The Reds won*" entails "*Either the Giants won or the Reds won*"

Inference Procedure

- KB $\vdash_i \alpha$
  - "α is derived from KB by inference algorithm *i*"
- Record of inference procedure operations is called a <u>proof</u>
- Inference procedure is <u>complete</u> if can find proof for any sentence that is entailed

Entailment and Inference

- Consider KB as a "haystack" and α as a "needle"
- **Entailment** is like the needle "being" in the haystack
- **Inference** is like "finding" the needle in the haystack

Inference

- Sentence is valid iff it is true under all possible interpretations in all possible worlds
  - Also called tautologies
  - "There is a stench at (1,1) or there is not a stench at (1,1)"
  - "There is an open area in front of me" is not valid in all worlds
- Sentence is satisfiable iff there is some interpretation in some world for which it is true
  - "There is a wumpus at (1,2)" could be true in some situation
  - "There is a wall in front of me and there is no wall in front of me" is unsatisfiable

Summary

- Intelligent agents need knowledge about the world and a means to reach good decisions
  - Representation of knowledge
  - Reasoning processes acting on knowledge
- Knowledge is contained in the form of sentences in a knowledge representation language that are stored in a knowledge base (KB)
- Representation language defined by syntax and semantics
  - Structure and meaning
- Inference is deriving new sentences from old ones