**The laboratory work 14**

**Wumpus world**

To run the presented generated world, download the Python 2.7.14 compiler <https://www.python.org/download/releases/2.7/>

Create a Python console application in Visual Studio. Create other files where you locate the code for generating a new world, and matching and instantiating new facts. In the main file we write the following facts and rules.

|  |
| --- |
| import updatewumpusNowWithRocks as wumpusimport FOPC as matcherfrom copy import deepcopy#initialization of the World where we playworld = wumpus.intialize\_my\_world('Cell 23', 'Cell 42', ['Cell 13', 'Cell 22', 'Cell 43'])#facts and rules that we will use to make inferencesfacts = []rules = [ ([('clean', '?x'), ('next', '?x', '?y')], [('no\_wumpus', '?y')]), ([('calm', '?x'), ('next', '?x', '?y')], [('solid', '?y')]), ([('no\_wumpus', '?x'), ('solid', '?x')], [('safe', '?x'), ]), ([('next', '?a', '?b'), ('next', '?b', '?c'), ('next', '?c', '?d'), ('next', '?d', '?a'), ('nasty', '?a'), ('no\_wumpus', '?b'), ('nasty', '?c')], [('wumpus', '?d'), ])]# Perceptions are our feelings in the cell where we standperceptions = []# These variables define whether it is our first step and whether we are stuck is\_first\_action = Trueis\_stuck = False# Defines whether we have rockshas\_rock = Truedef solve(): while not terminate(): take\_action() perceive() infer()# Perceive the current cell def perceive(): smell = perceptions[0] air = perceptions[1] glitter = perceptions[2] cur\_cell = perceptions[5] assert\_fact((smell, cur\_cell)) assert\_fact((air, cur\_cell)) assert\_fact((glitter, cur\_cell)) assert\_fact(('visited', cur\_cell)) if 'living' in perceptions: assert\_fact(('safe', cur\_cell)) next\_cells = wumpus.look\_ahead(world) for next\_cell in next\_cells: assert\_fact(('next', cur\_cell, next\_cell)) assert\_fact(('next', next\_cell, cur\_cell)) if not ask([('visited', next\_cell)]): assert\_fact(('unvisited', next\_cell))# Adding new factsdef infer(): while 1: new\_facts = [] for rule in rules: all\_bindings = [] match(rule[0], {}, all\_bindings) for bindings in all\_bindings: for conclusion in rule[1]: new\_facts.append(tuple(matcher.instantiate(conclusion, bindings))) if True not in map(lambda new\_fact: assert\_fact(new\_fact), new\_facts): breakdef ask(conditions): all\_bindings = [] match(conditions, {}, all\_bindings) return all\_bindingsdef match(conditions, bindings, all\_bindings): if bindings == False: return if not conditions: # different variable should have different value if len(set(map(lambda k: bindings[k], bindings))) == len(bindings): all\_bindings.append(bindings) return if conditions[0][-1] == '!': for fact in facts: if matcher.match(fact, conditions[0], deepcopy(bindings)) != False: return match(conditions[1:], bindings, all\_bindings) else: for fact in facts: new\_bindings = matcher.match(fact, conditions[0], deepcopy(bindings)) match(conditions[1:], new\_bindings, all\_bindings)# Asserting factsdef assert\_fact(new\_fact): if new\_fact in facts: return False # find conflictive fact for the same cell and remove it first facts\_to\_remove = [] for fact in facts: if is\_conflict(fact[0], new\_fact[0]) and fact[1] == new\_fact[1]: facts\_to\_remove.append(fact) for fact in facts\_to\_remove: facts.remove(fact) facts.append(new\_fact) return True# Checking different conflicts. Each cell can contain only one state for smell, air, gold, wumpus.def is\_conflict(val1, val2): s = set([val1, val2]) return s in [ set(['clean', 'nasty']), set(['calm', 'breeze']), set(['wumpus', 'no\_wumpus']), set(['visited', 'unvisited']), set(['bare', 'glitter']), ]# Here we do only one action depending on the cell we stand# 1. If in line of wumpus, shoot the arrow to kill it# 2. Move to a cell that's in line of wumpus when knowing its position, inorder to kill it# 3. Move to an unvisited safe cell# 4. Do nothing but perceive laterdef take\_action(): global perceptions global is\_first\_action global is\_stuck global has\_rock if is\_first\_action: perceptions = wumpus.take\_action(world, 'Down') cur\_cell = perceptions[5] assert\_fact(('home', cur\_cell)) assert\_fact(('visited', cur\_cell)) is\_first\_action = False return cur\_cell = perceptions[5] if ask([('glitter', cur\_cell)]): perceptions = wumpus.take\_action(world, 'PickUp') assert\_fact(('bare', cur\_cell)) elif ask([('wumpus', '?x')]): # shoot an arrow to the wumpus wumpus\_cell = ask([('wumpus', '?x')])[0]['?x'] orientation = face\_to(cur\_cell, wumpus\_cell) wumpus.take\_action(world, orientation) perceptions = wumpus.take\_action(world, 'Shoot') assert\_fact(('no\_wumpus', wumpus\_cell)) assert\_fact(('solid', wumpus\_cell)) for cell in get\_adjacent\_cells(wumpus\_cell): assert\_fact(('clean', cell)) elif ask([('unvisited', '?x'), ('safe', '?x')]): # move towards the unvisited and safe cell next\_cell = next\_cell\_to([('unvisited', '?x')], '?x') orientation = face\_to(cur\_cell, next\_cell) wumpus.take\_action(world, orientation) perceptions = wumpus.take\_action(world, 'Step') elif has\_rock and ask([('breeze', cur\_cell), ('checked', cur\_cell, '!'), ('next', cur\_cell, '?x'), ('unvisited', '?x'), ('pit', '?x', '!'), ('solid', '?x', '!')]): # toss a rock to a suspect cell suspect\_cells = ask([('next', cur\_cell, '?x'), ('unvisited', '?x'), ('pit', '?x', '!'), ('solid', '?x', '!')]) if not suspect\_cells: assert\_fact(('checked', cur\_cell)) return suspect\_cell = suspect\_cells[0]['?x'] orientation = face\_to(cur\_cell, suspect\_cell) wumpus.take\_action(world, orientation) sound = wumpus.take\_action(world, 'Toss') if sound == 'Quiet': assert\_fact(('pit', suspect\_cell)) elif sound == 'Clink': assert\_fact(('solid', suspect\_cell)) else: has\_rock = False elif has\_rock and ask([('breeze', '?x'), ('checked', '?x', '!'), ('next', '?x', '?y'), ('unvisited', '?y'), ('pit', '?y', '!'), ('solid', '?y', '!')]): # move to a unchecked cell that is not nasty but breezing next\_cell = next\_cell\_to([('breeze', '?x'), ('checked', '?x', '!'), ('next', '?x', '?y'), ('unvisited', '?y'), ('pit', '?y', '!'), ('solid', '?y', '!')], '?x') orientation = face\_to(cur\_cell, next\_cell) wumpus.take\_action(world, orientation) perceptions = wumpus.take\_action(world, 'Step') elif enough\_score(): # move to home or exit if ask([('home', cur\_cell)]): perceptions = wumpus.take\_action(world, 'Exit') return home\_cell = next\_cell\_to([('home', '?x')], '?x') orientation = face\_to(cur\_cell, home\_cell) wumpus.take\_action(world, orientation) perceptions = wumpus.take\_action(world, 'Step') else: is\_stuck = True# The game is over when we are dead or we have found the gold and come to Cell-11def terminate(): return is\_stuck or 'dead' in perceptions or 'won' in perceptions# Finding the location of Wumpusdef find\_wumpus(): wumpus\_cell = get\_wumpus\_loc() if not wumpus\_cell: return False cur\_cell = perceptions[5] return cur\_cell[5] == wumpus\_cell[5] or cur\_cell[6] == wumpus\_cell[6]# Checking whether we have enough points to move to the exitdef enough\_score(): return perceptions and perceptions[-1] > 0# Defining a new cell to movedef next\_cell\_to(conditions, var): cur\_cell = perceptions[5] q = [cur\_cell] parent = {} visited = set() cell = None while q: cell = q.pop(0) if ask(map(lambda c: tuple(matcher.instantiate(c, {var: cell})), conditions)): break all\_bindings = ask([('next', cell, '?x'), ('safe', '?x')]) next\_cells = map(lambda bindings: bindings['?x'], all\_bindings) for next\_cell in next\_cells: if next\_cell not in visited: visited.add(next\_cell) parent[next\_cell] = cell q.append(next\_cell) while parent[cell] != cur\_cell: cell = parent[cell] return cell# Defining in what direction we turneddef face\_to(from\_cell, to\_cell): from\_cell = perceptions[5] # `Cell 12` and `Cell 11` if from\_cell[5] == to\_cell[5]: return 'Up' if to\_cell[6] > from\_cell[6] else 'Down' elif from\_cell[6] == to\_cell[6]: return 'Right' if to\_cell[5] > from\_cell[5] else 'Left' return Nonedef get\_adjacent\_cells(cell): all\_bindings = ask([('next', cell, '?x')]) return map(lambda bindings: bindings['?x'], all\_bindings) solve() |