Algorithm 1 Search Function

Require: CROSS MOVES ← [(1,0),(-1,0),(0,1),(0,-1)]
Require: DIAGONAL MOVES ← [(1,1),(-1,-1),(1,-1),(-1,1)]
Require: a KnowledgeBase

origin ← (0,0)
base ← origin
loop
  base state ← **perceive**
  KnowledgeBase.**put**(base state)
  infer(base)
  explored(base.X,base.Y) ← *true*

  MOVES ← CROSS MOVES ∪ **shuffle**(DIAGONAL MOVES)
  for all Move m in MOVES **do**
    newX ← base.x + m.x
    newY ← base.y + m.y
    if isSafe(newX,newY) ∧ ¬ **explored**(newX,newY) **then**
      moveTo(newX,newY)
      state ← **perceive**
      KnowledgeBase.**put**(state)
      infer(state)
      explored(newX,newY) ← *true*
      if isGlittery(state) **then**
        pickUpGold
        escape(newX,newY)
      end if
      if m is the last Move in MOVES ∧ ¬**isBlack**(state) **then**
        base ← (newX,newY)
      else
        moveTo(base.x,base.y)
      end if
    end if
  end for
end loop

Algorithm 2 Inference Function

Require: a state to infer knowledge about
Require: CROSS MOVES ← [(1,0),(-1,0),(0,1),(0,-1)]
Require: a KnowledgeBase to update

```
1:  for all Move m in CROSS MOVES do
2:    adjacentX ← state.x + m.x
3:    adjacentY ← state.y + m.y
4:    if KnowledgeBase.contains(adjacentX,adjacentY) then
5:      adjState ← KnowledgeBase.get(adjacentX,adjacentY)
6:    else
7:      adjState ← new State
8:    end if
9:
10:   if isEmpty(state) then
11:     adjState.isEmpty ← true
12:   else
13:     if isBreezy(state) then
14:       adjState.pitPossibility ← adjState.pitPossibility + 1
15:     end if
16:     if isSmelly(state) then
17:       adjState.wumpusPossibility ← adjState.wumpusPossibility + 1
18:     end if
19:   end if
20:   KnowledgeBase.update(adjState)
21: end for
```

Algorithm 3 Safety Evaluation Function

Require: a position (x,y) to evaluate
Require: a KnowledgeBase

```
1: state ← KnowledgeBase.get(x,y)
2: if isEmpty(state) ∨ (state.pitPossibility = 0 ∧ state.wumpusPossibility
   = 0) then
3:   return true
4: else
5:   return false
6: end if
```

Algorithm 4 Escaping Function

Require: a KnowledgeBase
Require: a startingPosition
1: currentPosition ← startingPosition
2: **repeat**
3: nextPosition ← a safe neighbour of currentPosition that minimises the straight line distance to (0,0)
4: **moveTo**(nextPosition)
5: currentPosition ← nextPosition
6: **until** currentPosition = (0,0)