

Below are the algorithms that make up the AI for our agent. We chose to use a model-based agent, as a purely reflex based agent would not take advantage of the possibilities of logical inference in the world, and the lack of information about the proximity of the gold would have made a goal-based agent useless.

The agent only moves to tiles that it knows are safe, and that it has not visited previously. If the agent ever finds itself in a position in which it cannot move to a safe, unexplored tile, it will return to the origin and start over. This means that we reduce the risk of our agent trapping itself in a region of the map, but also that our agent refuses to accept the possibility that there is no way of reaching the gold (if any exists). Our AI will *never give up*.

List of Algorithms

1	Search Function - explore()	2
2	Inference Function - infer()	3
3	Safety Evaluation Function - isSafe()	4
4	Escaping Function - escape()	4

Algorithm 1 Search Function - **explore()**

Require: CROSS MOVES $\leftarrow [(1,0),(-1,0),(0,1),(0,-1)]$ **Require:** DIAGONAL MOVES $\leftarrow [(1,1),(-1,-1),(1,-1),(-1,1)]$

```
1: KnowledgeBase  $\leftarrow \emptyset$ 
2: explored  $\leftarrow \emptyset$ 
3: origin  $\leftarrow (0,0)$ 
4: base  $\leftarrow$  origin
5: loop
6:   if explored(base.X,base.Y) then
7:     escape(base.X,base.Y)
8:   else
9:     base state  $\leftarrow$  perceive
10:    KnowledgeBase.put(base state)
11:    infer(base)
12:    explored(base.X,base.Y)  $\leftarrow true$ 
13:
14:    MOVES  $\leftarrow$  CROSS MOVES  $\cup$  shuffle(DIAGONAL MOVES)
15:    for all Move m in MOVES do
16:      newX  $\leftarrow$  base.x + m.x
17:      newY  $\leftarrow$  base.y + m.y
18:      if isSafe(newX,newY)  $\wedge \neg$  explored(newX,newY) then
19:        moveTo(newX,newY)
20:        state  $\leftarrow$  perceive
21:        KnowledgeBase.put(state)
22:        infer(state)
23:        explored(newX,newY)  $\leftarrow true$ 
24:        if isGlittery(state) then
25:          pickUpGold
26:          foundTheGold  $\leftarrow true$ 
27:          escape(newX,newY)
28:        end if
29:        if m is the last Move in MOVES  $\wedge \neg$ isBlack(state) then
30:          base  $\leftarrow$  (newX,newY)
31:        else
32:          moveTo(base.x,base.y)
33:        end if
34:      end if
35:    end for
36:  end if
37: end loop
```

Algorithm 2 Inference Function - **infer()**

Require: a state to infer knowledge about

Require: CROSS MOVES $\leftarrow [(1,0),(-1,0),(0,1),(0,-1)]$

Require: a KnowledgeBase to update

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

Algorithm 3 Safety Evaluation Function - **isSafe()**

Require: a position (x,y) to evaluate

Require: a KnowledgeBase

```
1: state  $\leftarrow$  KnowledgeBase.get(x,y)
2: if isEmpty(state)  $\vee$  (state.pitPossibility = 0  $\wedge$  state.wumpusPossibility = 0) then
3:   return true
4: else
5:   return false
6: end if
```

Algorithm 4 Escaping Function - **escape()**

Require: a KnowledgeBase

Require: a startingPosition

Require: whether we foundTheGold

```
1: currentPosition  $\leftarrow$  startingPosition
2: repeat
3:   nextPosition  $\leftarrow$  a safe neighbour of currentPosition that minimises
   the straight line distance to (0,0)
4:   moveTo(nextPosition)
5:   currentPosition  $\leftarrow$  nextPosition
6: until currentPosition = (0,0)
7: if  $\neg$ foundTheGold then
8:   explore
9: end if
```
