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

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

```
Require: CROSS MOVES ← [(1,0),(-1,0),(0,1),(0,-1)]
Require: DIAGONAL MOVES ← [(1,1),(-1,-1),(1,-1),(-1,1)]
Require: KnowledgeBase ← ∅
Require: explored ← ∅
origin ← (0,0)
base ← origin
loop
  if explored(base.X,base.Y) then
    escape(base.X,base.Y)
  else
    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
          foundTheGold ← true
          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 if
end loop
```

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

```
Require: a state to infer knowledge about
Require: CROSS MOVES ← [(1,0),(-1,0),(0,1),(0,-1)]
Require: a KnowledgeBase to update
for all Move m in CROSS MOVES do
  adjacentX ← state.x + m.x
  adjacentY ← state.y + m.y
  if KnowledgeBase.contains(adjacentX,adjacentY) then
    adjState ← KnowledgeBase.get(adjacentX,adjacentY)
  else
    adjState ← new State
  end if

  if isEmpty(state) then
    adjState.isEmpty ← true
  else
    if isBreezy(state) then
      adjState.pitPossibility ← adjState.pitPossibility + 1
    end if
    if isSmelly(state) then
      adjState.wumpusPossibility ← adjState.wumpusPossibility + 1
    end if
  end if
  KnowledgeBase.update(adjState)
end for
```

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

Require: a position (x,y) to evaluate
Require: a KnowledgeBase
state $\leftarrow$ KnowledgeBase.**get**(x,y)
if **isEmpty**(state) $\vee$ (state.pitPossibility = 0 $\wedge$ state.wumpusPossibility = 0) **then**
    return *true*
else
    return *false*
end if

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

Require: a KnowledgeBase
Require: a startingPosition
Require: whether we foundTheGold
currentPosition $\leftarrow$ startingPosition
repeat
    nextPosition $\leftarrow$ a safe neighbour of currentPosition that minimises the straight line distance to (0,0)
    moveTo(nextPosition)
    currentPosition $\leftarrow$ nextPosition
until currentPosition = (0,0)
if $\neg$foundTheGold **then**
    explore
end if