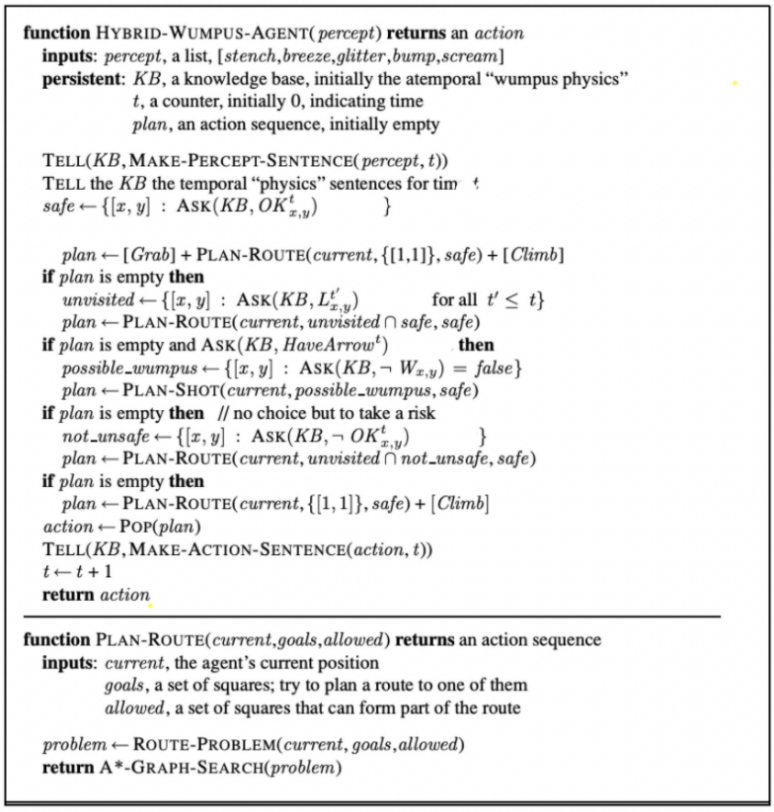
HW5 Part B

1. A hybrid model calculates the safety probability for all the squares that is not yet proved safe or unsafe after each move. It chooses the safest square if there are no unvisited safe square remaining.

Modified Code



function HYBRID-WUMPUS-AGENT(*percept*) **returns** an *action*
  inputs: *percept*, a list, [*stench*, *breeze*, *glitter*, *bump*, *scream*]
  persistent: KB, a knowledge base, initially the atemporal "wumpus physics"
    t, a counter, initially 0, indicating time
    plan, an action sequence, initially empty

  TELL(KB, MAKE-PERCEPT-SENTENCE(*percept*, t))
  TELL the KB the temporal "physics" sentences for tim t
  safe ← {[x, y] : ASK(KB, $OK^t_{x,y}$) }

  plan ← [*Grab*] + PLAN-ROUTE(*current*, {[1,1]}, *safe*) + [*Climb*]
  if *plan* is empty **then**
    unvisited ← {[x, y] : ASK(KB, $L^{t'}_{x,y}$) for all $t' \leq t$}
    plan ← PLAN-ROUTE(*current*, *unvisited* ∩ *safe*, *safe*)
  if *plan* is empty and ASK(KB, *HaveArrow*^{t}) **then**
    possible_wumpus ← {[x, y] : ASK(KB, ¬ $W_{x,y}$) = *false*}
    plan ← PLAN-SHOT(*current*, *possible_wumpus*, *safe*)
  if *plan* is empty **then** // no choice but to take a risk
    not_unsafe ← {[x, y] : ASK(KB, ¬ $OK^t_{x,y}$) }
    plan ← PLAN-ROUTE(*current*, *unvisited* ∩ *not_unsafe*, *safe*)
  if *plan* is empty **then**
    plan ← PLAN-ROUTE(*current*, {[1, 1]}, *safe*) + [*Climb*]
  action ← POP(*plan*)
  TELL(KB, MAKE-ACTION-SENTENCE(*action*, t))
  t ← t + 1
  return *action*

function PLAN-ROUTE(*current*, *goals*, *allowed*) **returns** an action sequence
  inputs: *current*, the agent's current position
    goals, a set of squares; try to plan a route to one of them
    allowed, a set of squares that can form part of the route

  problem ← ROUTE-PROBLEM(*current*, *goals*, *allowed*)
  return A*-GRAPH-SEARCH(*problem*)

function $ASK(KB, OK^t_{x,y})$
input: $P[x, y]$ The safe probability
$P[x, y] = p(p_1 \dots p_n)//$product rule
return $Max(P)$

Program Modifications

```
def PitWumpus_probability_distribution(self, width, height):
    fringe = []
```

```
    fringe = list(self.available_rooms)
    known_BS = self.observation_breeze_stench(self.visited_rooms)
    known_PW = self.observation_pits(self.visited_rooms)

    P = JointProbDist(fringe, { each:[T, F] for each in fringe })

    events = all_events_jpd(fringe, P, known_PW)

    room_size = self.cave.WIDTH * self.cave.HEIGHT

    p_true = 0.2
    p_false = 0.8
    for each in events:
        true_count = 0
        for v in each.values():
            if v:
                true_count += 1
        P[each] = (p_true ** true_count) * (p_false ** (room_size - true_count))
    return P
```

In the original code, the probability was calculated using `breeze_stench`. The new way of calculating the probability is to use the surrounding square. If the surrounding blocks contain traps, the probability of safety is `0.2` else `0.8` .

> This results in a hybrid agent which takes lesser number of moves on average than a logical agent, less likely to end up with no more rooms to explore and also explores more possibilities than a typical probabilistic agent.
