Appendix D.
Pseudo-code

```
Create results, an object that stores the expected number of cues sampled
      for all possible environments and all possible starting budgets;
Initialize ENV, the set of all environments [section B.1.4]
Initialize S^start, the set of all possible states an agent can start a cycle
      with

FOR all environments env ∈ ENV:

   Determine the expected utility v_u^*(s) [section C.3]
   Set k to 0;
   Set δ to ∞;
   Set ϵ to 0.001;
   Set v_u^k(s) to 0 for each s ∈ S^start;

   WHILE δ > ϵ:
      Increment k with 1
      FOR all somatic states s ∈ S^start:
         Do a forward pass:
         Create a root node for s_start that has a starting somatic state of
            s;
         Iteratively expand the root node and all of its successors to
            create a tree Tree;

         Do a backward pass:
         Determine all transition states S^transition
         Set the expected utility U(s) of all transition states
            s_transition ∈ S^transition in Tree to v_u^{k-1}(s_transition);
         Use backwards induction to find the expected utility U(s) for all
            non-transition states in Tree;
         Set v_u^k(s) to U(s_start);
      END FOR
      Set δ to δ(v_u^{k-1}(s), v_u^k(s))
   END WHILE
   Set v_u^*(s) to v_u^k(s)

   Finding the optimal policy during first cycle [section C.2]
   Initialize v_f^*(s)
   FOR all s ∈ S^start
      Set v_f^*(s) to ω(v_u^*(s))
   FOR all somatic states s ∈ S^start:
      Do a forward pass:
            Create s_start, the root node;
            Iteratively expand the root node and all of its successors
                  create a tree Tree;

      Do a backward pass:
            Set the expected fitness of all final states s_final in Tree to
                  v_f^*(s);
            Use backwards induction to find the set of best actions A^*(s)
                  for all non-terminal states in T;

      Do a forward pruning pass:
            Create S_leaf, the set containing all leaf nodes in Tree;
```

```
            Compute for each state in Tree the proportion prop(s) of agents
                  that visit that state;
            For each leaf node, compute weightedCuesSampled(s) by multiplying
                  the expected number of cues sampled in that state with
                  that state's prop(s);

      Compute number, the expected number of sampled cues, by summing the
            weightedCuesSampled(s) of all leaf nodes;
      Store the expected number of cues sampled in results
   END FOR
END FOR
```