Programming Exercise for 2APL

This programming exercise will illustrate some of the features of 2APL through the development and refinement of the Harry and Sally scenario, whose implementation is taken as the starting point.

This exercise consists of three sub-exercises. In each of those sub-exercises, we start with a general description and the objective, followed by some brief explanation of the steps involved in achieving the exercise and some hints on how to do. Depending on your programming experience, you may decide to only read the introduction and objective to develop your own original solution, or also read the explanation and hints.

a) **Introduction**: Harry will have to share its environment with other agents specialized in picking up bombs and dropping them on the bomb traps. To know who drops more bombs on a trap, we will trust agents to keep note of how many bombs they have dealt with.

   Objective: modify harry.2apl specification file in a way that Harry knows, at each point of its execution, how many bombs he dropped on traps.

   Description: the objective can be accomplished by maintaining a belief whose argument is the number of bombs dealt with. You should start by creating an initial belief (e.g. **`score(0)`**). Then, this belief should be updated each time a bomb is dropped on a trap. This can be done as part of the specification of the belief update **`Drop()`**, assuming that your agent will only drop bombs at the right location, i.e. the bomb traps.

   Hints: to update the belief **`score(N)`** you can remove it and add **`score(N+1)`**. The predicate **`is(X,A+B)`** succeeds by instantiating **`X`** with the result of **`A+B`**.

b) **Introduction**: As you can see, the current implementation of Harry includes a belief (**`trap(0,0)`**) indicating the location of one bomb trap. We will want Harry to be able to perform in environments where there is no prior knowledge about the location of the bomb trap. In fact, there may even be more than one bomb traps.

   Objective: modify the current harry.2apl specification file in a way that Harry can operate in environments where there may be more than one bomb trap, whose locations are not known in advance.

   Description: the objective can be accomplished if Harry successfully finds at least one bomb trap before it starts cleaning the world. This can be accomplished e.g. through the definition of a goal (e.g. **`knowtrap`**) which is true whenever **`trap(X,Y)`** is true for some **`X,Y`**. Then, you will have to define a planning goal rule (PG rule) for the goal **`knowtrap`** which adds (through a belief update) the belief **`trap(X,Y)`** when a trap is found. Do not forget that Harry should not start cleaning the world before it knows of a bomb trap.

   Hints: to keep things simple, it may be easiest to adopt the plan used by Sally to find bombs, to allow Harry to find a trap. The external action **`@blockworld(senseTraps(),TRAPS)`** may prove handy. To avoid start cleaning the world before knowing a trap, it may be a good idea to extend the belief query (the guard) of the corresponding plan.

c) **Introduction**: When competing with other agents for the bigger number of bombs dealt with, it proves to be important to be near Sally when it finds bombs, since there is a race between agents to get there first. When Harry is busy cleaning the world, there isn't much that can be done (apart from maybe looking for other bomb traps so that the closer one can always be chosen). But when Harry is not doing anything, it may be a good strategy to simply follow Sally while waiting for the next bomb.

Objective: modify the current harry.2apl specification file in a way that Harry follows Sally in case it is not doing anything (i.e. it knows of a bomb trap and has dealt with all the bombs it knows).

Description: the objective can be accomplished through the definition of a plan that is executed when Harry is not doing anything which, in its simplest version, simply senses its surroundings for agents and, if Sally is in the vicinity, moves to its position.

Hints: the external action `@blockworld( senseAllAgent(), AGENTS )` returns all agents in the vicinity.