A SNePS Approach to The Wumpus World Agent or Cassie Meets the Wumpus

Stuart C. Shapiro and Michael Kandefer
Department of Computer Science and Engineering and Center for Cognitive Science
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000
{shapiro|mwk3}@cse.buffalo.edu

1 Introduction

We demonstrate the use of SNeRE, the acting component of the SNePS knowledge representation, reasoning, and acting system, by showing its use to implement a wumpus world agent [Russell and Norvig, 1995][1]. For this purpose, we use SNePS 2.6.2, which consists of SNePS 2.6.1 [Shapiro *et al.*, 2004] plus some patch files. We usually name our SNePS-based agents Cassie [Shapiro, 1989; 1998; Shapiro and Ismail, 2003; Shapiro *et al.*, 2000; Shapiro and Rapaport, 1987; 1991]. To distinguish Cassie in the role of the wumpus world agent, we will call her CassieW.

Our main motivation in developing intelligent systems is to model general human-level intelligence, not to maximize the use of computing power to optimize problem solving. CassieW has been developed accordingly.

2 The Wumpus World

The wumpus world consists of a rectangular world of cells, within which is a rectangular cave, the size of which can vary from run to run. The cells within the cave are considered to be rooms; the border of the cave is a wall formed by cells that are not rooms. Each cell is identified by its Cartesian coordinates, `cell(x,y)`, $-1 \leq x \leq max_x$, $-1 \leq y \leq max_y$, where max_x and max_y are parameters that are fixed within a set of runs. `cell(0,0)` is always in one corner of the cave, and is CassieW's "home room", where she starts, facing east.

Each room in the cave, other than `cell(0,0)`, has a 20% probability of containing a pit. A live wumpus and a bar of gold are also placed in the cave randomly. Neither can be in `cell(0,0)` nor in a room with a pit, although they can be in the same room as each other.

If CassieW ever goes into a room containing a pit or the live wumpus, she dies. However, in each of the rooms adjacent to a room that contains a pit, CassieW can detect a breeze, and in each of the rooms adjacent to a room that contains the wumpus, she can detect a stench. She can also detect a glitter in the room with the gold.

CassieW starts out with one arrow. If she shoots the arrow, it will travel in the direction she is facing until it either hits the wumpus or the far wall. If it hits the wumpus, the wumpus dies and CassieW can hear it scream.

[1] As described on `http://www.cl.inf.tu-dresden.de/~mit/LRAPP/wumpus/wumpus.htm`

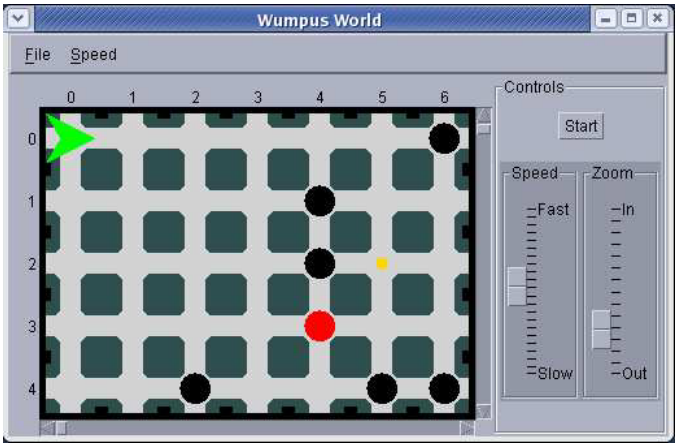

Figure 1: The Wumpus World. CassieW is in the upper, left-hand corner, facing east. The small yellow dot is the gold; the red circle southwest of the gold is the wumpus; the other (black) circles are the pits.

CassieW's task is to find the gold, grab it, return home (to `cell(0,0)`), and stop.

For the graphical aspects of CassieW and her wumpus world, we are using Byron Weber Becker's Java implementation of Rich Pattis' Karel the Robot[2]. Due to constraints of this system, `cell(0,0)` is in the north-west corner instead of its usual position in the south-west corner; `cell(1,0)` is to its east, and `cell(0,1)` to its south. Our wumpus world is shown in Fig. 1.

CassieW is capable of performing the following standard wumpus-world primitive acts.

- `go(`*d*`)`: If *d* is `left` or `right`, CassieW turns 90° left or right, respectively. If *d* is `forward`, CassieW goes to the room in front of her. However, if there is a wall in front of her, she doesn't move, but can detect that she has bumped into the wall.
- `do(grab)`: CassieW grabs for the gold; if she's in the room with the gold, she's successful.
- `do(shoot)`: If she still has her arrow, CassieW shoots it.

[2] *see* `http://www.learningwithrobots.com/`

- `do(stop)`: CassieW terminates all her activity; if she's at `cell(0,0)`, she exits the cave.
- `senseFor(`*`percept`*`)`: CassieW actively senses for one of the possible percepts: `stench`, `breeze`, or `glitter`. See §11 for how CassieW perceives bumps and screams.

CassieW can also `do(nothing)`, which is the act of not doing anything.

When CassieW either stops or dies, she receives a score, which is printed. The total score is the sum of: -1 for each `go(`*`d`*`)`, `do(grab)`, or `do(stop)`; -10 for `do(shoot)`; -1,000 for dying; and +1,000 for being in `cell(0,0)` with the gold when CassieW stops.

3 Use of The GLAIR Architecture

CassieW is implemented following the GLAIR (Grounded Layered Architecture with Integrated Reasoning) architecture [Hexmoor *et al.*, 1993; Hexmoor and Shapiro, 1997; Shapiro and Ismail, 2003], and uses the following layers.

The Knowledge Layer (KL) is the layer at which "conscious" reasoning takes place. The KL is implemented in SNePS [Shapiro, 2000; Shapiro *et al.*, 2004], and its subsystem SNeRE (the SNePS Rational Engine) [Kumar, 1996; Kumar and Shapiro, 1994a; 1994b; Shapiro *et al.*, 2004]. SNePS, in turn, is implemented in Common Lisp.

The Perceptuo-Motor Layer, Sublayer a (PMLa) Contains the Common Lisp implementation of the actions that are primitive at the KL. PMLa is implemented in a way that takes into account the top-level design of the agent, but is independent of the implementation of the agent's body.

The Perceptuo-Motor Layer, Sublayer b (PMLb) Implements the functions of PMLa taking into account the particular implementation of the agent's body and environment. CassieW's PMLb uses Franz Inc.'s Allegro CL jLinker[3] to link Common Lisp code to Java programs, in which the lower layers are implemented.

The Perceptuo-Motor Layer, Sublayer c (PMLc); The Sensori-Actuator Layer (SAL); and The Environment are implemented as a set of Java classes and methods that specialize the Java implementation of Karel the Robot, and which is responsible for the display in Fig. 1.

4 Some SNePS Basics

A SNePS knowledge base is seen as containing the beliefs of the agent, itself, rather than being information about the agent. In that sense, the SNePS KB contains *first-person* beliefs of the agent. Of course, an agent may have beliefs about other agents, and these nested beliefs can be represented in SNePS [Shapiro and Rapaport, 1991; Chalupsky and Shapiro, 1996], but this facility is not used for CassieW. Another aspect of first-person representation is that what is criterial for a belief's being in the KB is not that it is true in the world, but that the agent is justified in believing it.

Similarly, SNePS can be used to reason about the actions of other agents, but the primary use of the SNePS acting system, and the use presented in this paper, is for the agent, itself, to act: it is a first-person acting system. It is also an on-line acting system. That is, it's primary use, and the use presented in this paper, is to control the agent's current acting. CassieW acts in her world, and, when necessary, she reasons about what she should do next based on: her beliefs about the current state of the world; the evidence of her sensory apparatus; a set of small stored or inferred plans (recipes) for carrying out certain actions or for bringing about certain states. This sensing, reasoning, and inferring is done on-line, while CassieW is acting.

The contents of CassieW's KL (her beliefs) will be shown using SNePSLOG [Shapiro *et al.*, 2004], which is one of a set of interface languages used to interact with SNePS agents. The current SNePSLOG syntax does not allow a formula to be a simple atomic symbol. It must consist of at least one function or predicate symbol with at least one argument. For example, neither a proposition such as `HaveGold` nor an act such as `shoot` is legal. Instead, CassieW uses `Have(gold)` and `do(shoot)`, respectively.

In SNePS, propositions are reified [Shapiro, 1993]. That is, they are considered first-class members of the domain. So it is not really the case that `Have`, as used above, is a predicate symbol, nor that `Have(gold)` is a sentence denoting a truth value. Instead, `Have` is a function symbol, and `Have(gold)` is a proposition-valued functional term. Similarly, the SNePSLOG expression `Have(gold) and ~Alive(wumpus)` is a functional term denoting the proposition, *"I have the gold and the wumpus is not alive."* A SNePS agent may contemplate or have beliefs about propositions that it does not believe. If CassieW believes that she is facing east, we will say that she believes `Facing(east)`, or that `Facing(east)` is asserted. An explicit `Holds` predicate is neither needed nor used.

The designers of any reasoning system must face the issue of, when new information, p, is inferred, should it be saved in the knowledge base? This is a traditional space-time tradeoff. However, it may also be that p is a necessary step of a much longer derivation of q, and storing p may shorten that later derivation. Focussing on this role for p, we will refer to derived information that might or might not be saved in the knowledge base as **lemmas**. SNePS has been designed to save lemmas in the knowledge base.

Another basic decision for the designers of an agent is whether to give the agent a model of time. By this we mean whether the agent will have beliefs that certain events happened, or that certain acts were performed, at certain times. Some previous versions of Cassie (*e.g.*, [Shapiro, 1998; Ismail and Shapiro, 2000; Ismail, 2001; Shapiro and Ismail, 2003]) had models of time. The alternative to a model of time is to have the agent have only situation-independent beliefs and beliefs about the current situation. These can, of course, include beliefs about past events and acts as long as multiple past times needn't be distinguished. CassieW does not have a model of time, but can believe propositions such as `Visited(cell(2,3))`, meaning "I have visited cell (2,3)."

The belief that a fluent, a situation-dependent proposition, held at a paticular time, once believed, may remain believed. However, the belief that a fluent holds *now* must be disbe-

[3] See http://www.franz.com/support/documentation/7.0/doc/

lieved once it no longer holds. If any lemmas were derived from such fluents, they must be disbelieved also. SNePS uses SNeBR [Martins and Shapiro, 1988], an assumption-based truth maintenance system, for such house-cleaning.

Since the developers of SNePS have been interested in modeling general human-level intelligence, we have not built any numerical processing into SNePS. Therefore CassieW has been given the explicit beliefs `Isa(`*i*`, Number)`, $-1 \leq i \leq max(max_x, max_y)$, and `Successor(`*i*`, `*i+1*`)`, $-1 \leq i < max(max_x, max_y)$.

5 Directions and State Constraints

CassieW has a sense of direction, for which she uses: the individual constants, `north`, `south`, `east`, and `west` denoting the four directions; the individual constant `Direction` denoting the category of directions; and the proposition `Isa({north,south,east,west}, Direction)` denoting the proposition that north, south, east, and west are directions. The proposition that each of `north`, `south`, `east`, and `west` is a `Direction` follows from this by the SNePS method of reduction inference [Shapiro, 1991].

CassieW also needs to know how the directions are arranged around the compass, for which she uses: `Clockwise(`*d1,d2*`)` for the proposition that direction *d2* is clockwise from direction *d1*. CassieW believes `Clockwise(north, east)`, `Clockwise(east, south)`, `Clockwise(south, west)`, and `Clockwise(west, north)`.

CassieW needs to know what direction she's facing. For this, she uses the proposition `Facing(`*d*`)`, for the proposition, "I am facing direction *d*". as well as the belief that she's always facing in exactly one direction: `andor(1,1){Facing(north), Facing(south), Facing(east), Facing(west)}`. In SNePSLOG, `andor(i,j){`*P1, ..., Pn*`}`, where {*P1, ..., Pn*} is a set of propositions, denotes the proposition that at least *i* and at most *j* of the n *Pk* are true. So `andor(1,1){`*P1, ..., Pn*`}` denotes the proposition that exactly one of the *Pk* is true, and constitutes a **state constraint**. The use of state constraints will be discussed in §8. Note also that `andor(0,0)` is generalized nor, and $\sim p$ is an abbreviation of `andor(0,0){`*p*`}`. At the beginning, CassieW believes that `Facing(east)`, from which it follows that she isn't facing north, south, or west.

6 The Cells and Rooms

Each cell in the wumpus world is denoted by the functional term `cell(`*x,y*`)`. At the beginning, CassieW is given the beliefs that `Isa(cell(`*x,y*`), Room)`, $\sim$`Isa(cell(`*x,-1*`), Room)`, and $\sim$`Isa(cell(-1,`*y*`), Room)`, where $0 \leq x \leq max_x$, $0 \leq y \leq max_y$, and `Room` is an individual constant denoting the category of rooms in the cave, A cell that is not a room is part of the wall surrounding the cave, so CassieW starts off knowing where the north and west walls are. She will have to discover where the south and east walls are by herself.

CassieW is also given complete adjacency information using `Adjacent3(`*c1,c2,d*`)` for the proposition that cell *c2* is *d*-of cell *c1*. That is, CassieW believes that each room in the cave, including rooms next to walls, is adjacent to four cells.

Sometimes it is sufficient for CassieW to know that two rooms are adjacent without thinking about which direction one is from the other. For this, she uses the proposition `Adjacent(`*c1,c2*`)`, for the proposition that cell *c1* is adjacent to cell *c2*. `Adjacent(`*c1,c2*`)` is derivable from `Adjacent3(`*c1,c2,d*`)` by reduction inference.

CassieW is always in some cell (that's a room). Her belief that she's in cell *c* is represented by `In(`*c*`)`. CassieW is initialized with the belief that `In(cell(0,0))`, and the state constraint that `andor(1,1){..., In(cell(x,y)), ...}`, for $0 \leq x \leq max_x, 0 \leq y \leq max_y$.

Each room can also contain the wumpus or a pit. Of course, some room contains the gold, but CassieW never reasons about that — when she detects a glitter, she just grabs the gold. For the belief that a particular room contains the wumpus or a pit, CassieW uses `Contains(`*r,x*`)`, denoting the proposition that room *r* contains *x*.

For the second argument of `Contains`, we use one of the individual constants, `wumpus` or `pit`. Although `wumpus` denotes the one and only individual wumpus, there is no need to individuate particular pits, so `pit` is actually being used like a mass noun — one individual constant for all the pits. One might read `Contains(cell(3,5), pit)` as "*Cell (3,5) contains pit.*" At the start, CassieW believes that $\sim$`Contains(cell(0,0), wumpus)` and $\sim$`Contains(cell(0,0), pit)`.

For conciseness, we also use `Safe(`*c*`)` for the proposition, "cell *c* is safe for me to enter". Safety and containing a pit or the wumpus are connected by the beliefs that `all(c)(`$\sim$`Contains(c,pit) => ({`$\sim$`Alive(wumpus), `$\sim$`Contains(c,wumpus)} v=> {Safe(c)}))`, and `all(c)({Safe(c)} v=> {`$\sim$`Contains(c, pit), andor(1,2){`$\sim$`Contains(c, wumpus), `$\sim$`Alive(wumpus)}})`, where `Alive(wumpus)`, means that the wumpus is alive. In SNePSLOG, {A1, ..., An} v=> {C1, ..., Cm} means that if any *Ai* is believed, then any *Cj* may be believed.

Recall that all the rooms in the cave are cells, and have four adjacent cells. CassieW distinguishes rooms from walls by believing that each room is a cell, `c`, for which `Isa(c,Room)`, but that each wall-cell is a cell, `c`, for which $\sim$`Isa(c,Room)`. She never knowingly goes into a wall-cell, but she does believe that `all(c)(`$\sim$`Isa(c, Room) => Safe(c))`, which helps her locate the pits and the wumpus.

7 Propositions, Acts, and Policies

SNeRE recognizes three particular types of domain entities: propositions, acts, and policies. Propositions are entities that can be believed and whose negations can be believed. Acts are entities that a SNeRE agent can perform. Policies connect propositions and acts. Two SNeRE built-in policies are used by CassieW:

> `whendo(`*p,a*`)`: When I believe the proposition *p*, I will perform the act *a*.

`whenever do(p,a)`: Whenever I believe the proposition *p*, I will perform the act *a*.

In each case, if the policy has been adopted, the agent performs *a* when forward inference causes *p* to be believed. Also *a* is performed if *p* is already believed when the policy is adopted with forward inference. The difference is that a `whendo` policy is unadopted after firing once, but a `whenever do` remains adopted until explicitly unadopted.

We call something that the agent can perform an **act**. An act consists of an **action** and zero or more arguments. For example, CassieW's act of going one cell forward, expressed in SNePSLOG as `go(forward)`, consists of the action of going (`go`) and the argument forward (`forward`).

Since the smallest well-formed SNePSLOG expression is a functional term consisting of a function symbol and at least one argument, we use the functional term `do(a)` to represent the act of performing the action *a* on no arguments.

Any agent has a repertoire of primitive actions it can perform. We will say that an act whose action is a primitive action is a primitive act. We will call other acts and actions complex.

SNeRE comes with a set of preprogrammed primitive actions: mental actions, discussed in the next section, and control actions. The control actions used by CassieW (for the complete set, see [Shapiro *et al.*, 2004]) are:

- `do-all({a1, ..., an})`: Perform all the acts `a1, ..., an` in random order.
- `do-one({a1, ..., an})`: Perform one of the acts `a1, ..., an` chosen randomly.
- `prdo-one({pract(x1, a1), ..., pract(xn, an)})`: Perform one of the acts `aj`, with probability `xj/(x1 + ... + xn)`.
- `snif({if(p1,a1),...,if(pn,an)[, else(da)]})`: Using backward inference, determine which of the `pi` hold. If any do, randomly choose one of them, say `pj`, and perform `aj`. If none of the `pi` can be inferred, and if `else(da)` is included, perform `da`. Otherwise, do nothing.
- `snsequence(a1, a2)`: Perform `a1`, and then perform `a2`. For sequences of three acts, `snsequence3(a1,a2,a3)` is used.
- `withsome(?x, p(?x), a(?x), da)`: Using backward inference, determine which, if any, entities satisfy the open proposition `p(?x)`. If any do, randomly choose one, say `e`, and perform `a(e)`. If no entity satisfies `p(?x)`, perform `da`. `withsome/3` is like `withsome`, but with `da` defaulting to do nothing. `withall/3` is like `withsome/3`, but performs `a(e)` on all `e` that satisfy `p(?x)`

Additional primitive acts must be defined by the agent designer, and implemented in the PML and SAL. The primitive acts used by CassieW were described in §2.

8 Mental Acts

The two mental actions are `believe` and `disbelieve`.

When a SNeRE agent performs the mental act `disbelieve(p)`, the result is that if *p* is a believed proposition, it is no longer believed, and if *p* is an adopted policy, it is no longer adopted. Note that disbelieving a proposition does not cause its negation to be believed, and that when *p* is disbelieved, SNeBR causes any lemmas that depended on *p* to also be disbelieved (see §4).

When a SNeRE agent performs the mental act `believe(p)`, the result is: if *p* is a policy, it is adopted; if *p* is a proposition, it is believed as an hypothesis; and, in either case, forward inference is done with *p*. The forward inference may cause other propositions to be believed, policies to be adopted, and adopted policies to trigger.

However, before `believe` changes the belief status of a proposition *p*, it performs a limited form of belief revision[4] [Alchourrón *et al.*, 1985]:

1. If `andor(0,0)`{`...,p,...`} is believed, `disbelieve` it;
2. If `andor(i,1)`{`p,q,...`} and *q* are believed, disbelieve *q*.

Case 2 is the way state constraints (§5) are used.

9 `(p => q)` vs. `whendo(p,believe(q))`

When CassieW enters a room, r, she wants to remember that she has visited it, using the proposition `Visited(r)`, and that it's safe. One might think that this could be done with the rule `all(r)(In(r) =>` {`Visited(r), Safe(r)`}). However, in that case, after being in `cell(2,3)`, for example, `Visited(cell(2,3))` and `Safe(cell(2,3))` would be lemmas supported by `In(cell(2,3))`, and as soon as CassieW moved to another room, they would no longer be believed. Instead, this is represented by

```
all(r)(Isa(r, Room) => whendo(In(r),
            believe({Visited(r), Safe(r)})))
```

In that way, when `Visited(cell(2,3))` and `Safe(cell(2,3))` are believed, they are asserted as hypotheses, and remain so even when CassieW moves to another room. She starts off believing that `Visited(cell(0,0))`, and ∼`Visited(c)`, for all other cells, `c`.

10 The SNeRE Execution Cycle

An abbreviated[5] version of the SNeRE execution cycle makes use of these predefined proposition-forming functions:

- `ActPlan(a1,a2)`: A plan for performing the complex act *a1* is to perform the complex act *a2* (which is usually structured using one or more of the SNeRE control acts);
- `Effect(a,p)`: An effect of performing the act *a* is that the proposition *p* (which could be of the form ∼`q`) will hold.

The abbreviated execution cycle is:

To perform the act *a*:
  Use backward inference to find propositions *pe* such that `Effect(a,pe)`;
  if *a* is primitive, execute its implementation;
  else Use backward inference to find acts *a2* such that `ActPlan(a,a2)`, and perform one of them;
  for all *pe*, perform `believe(pe)`.

[4] We intend to extend this to a more unrestricted form of belief revision in the future.

[5] We ignore preconditions in this paper, since CassieW doesn't use them.

```
wheneverdo(Feel(breeze),
  withsome/3(?r, In(?r), believe(nexists(1,4,4)(c)({Adjacent(?r,c)}:{Contains(c,pit)}))))
wheneverdo(~Feel(breeze),
  withsome/3(?r,In(?r), withall/3(?c, Adjacent(?r,?c), believe(~Contains(?c,pit)))))
wheneverdo(Smell(stench),
  withsome/3(?r,In(?r), believe(nexists(1,1,4)(c)({Adjacent(?r,c)}:{Contains(c,wumpus)}))))
wheneverdo(~Smell(stench),
  withsome/3(?r,In(?r), withall/3(?c,Adjacent(?r,?c), believe(~Contains(?c,wumpus)))))
```

Figure 2: Policies for what to do when CassieW senses the presence or absence of a breeze or a stench.

```
wheneverdo(Feel(bump),
  snif({if(Facing(north), do(nothing)), if(Facing(west), do(nothing)),
        else(do-all({
          withsome/3(?x, In(cell(?x,?x)), withsome/3(?d, Facing(?d), assertWall(?d,?x))),
          withsome/3({?x,?y}, In(cell(?x,?y)), snif({if(Facing(east),assertWall(east,?x)),
                                                     else(assertWall(south,?y))})})})})
```

Figure 3: CassieW's policy when feeling a bump.

CassieW is naive enough to believe, as indicated here, that all her acts are effective. Other versions of Cassie have believed the effects of their acts only when they sense them.

11 Active and Passive Perception

Perception is accomplished in GLAIR agents by the PML performing a `believe` on a proposition that some object or phenomenon has been perceived. As mentioned in §8, this could cause inferences to be drawn and acts to be performed via adopted policies.

Active perception is accomplished by a sensory act, whose performance leads to a perception. Active perception is done by CassieW with the sensory acts `senseFor(stench)`, `senseFor(breeze)`, and `senseFor(glitter)`. They are combined into one complex act,

```
ActPlan(do(perceive), do-all(
 {snif(if(~Have(gold), senseFor(breeze))),
  snif(if(~Have(gold), senseFor(glitter))),
  snif(if(Alive(wumpus), senseFor(stench)))}))
```

so that she bothers to `senseFor(stench)` only if she believes that the wumpus is still alive, and to `senseFor(glitter)` and `senseFor(breeze)` only if she doesn't already have the gold. (As we'll see in §15, she doesn't have to worry about the pits when she is on her way home with the gold.) She starts out believing that `Alive(wumpus)` and ∼`Have(gold)`.

Upon performing `senseFor(`*x*`)`, where *x* is either `breeze` or `stench`, CassieW's PMLa performs a `believe` either on `Feel(`*x*`)` or ∼`Feel(`*x*`)`. These trigger the policies shown in Fig. 2, and allow CassieW to eventually locate the pits and the wumpus, if she explores the cave sufficiently. The formula `nexists(i,j,k)(x)({P(x)}:{Q(x)})` denotes the proposition that, of the `k` individuals `a` that satisfy `P(a)`, at least `i` and at most `j` also satisfy `Q(a)` [Shapiro, 1979].

Upon performing `senseFor(glitter)`, CassieW's PMLa performs a `believe` either on `See(glitter)` or ∼`See(glitter)`. Not seeing glitter is not noteworthy, but seeing glitter triggers the policy `whendo(See(glitter), do(grab))`, so that CassieW gets the gold as soon as she sees it. Then she believes that she has it, because of her belief that `Effect(do(grab), Have(gold))`.

The act `do(perceive)` is used in two other plans: first, since it needs to be done whenever CassieW goes into a new room, it is used in the plan `ActPlan(move(forward), snsequence(go(forward), do(perceive)))` and `move(forward)` is used in most other plans instead of `go(forward)`; and second, it is used in the plan for `get(gold)`, `ActPlan(get(gold), snsequence(do(perceive), explore(cave)))`, which is the top-level act CassieW is asked to perform.

Passive perception happens when the PML triggers a perception not in response to a sensory act. This is the way CassieW senses bumps and the wumpus' scream. She reacts to the scream according to the policy `whendo(Hear(scream), believe(∼Alive(wumpus)))`. She reacts to a bump according to the policy shown in Figure 3. This policy will cause CassieW to identify all the cells in the south and east walls when she first bumps into them. (She will only bump into the north or west walls when moving randomly (see §13).) The two `withsome/3` instances are needed because the SNePS Unique Variable Binding Rule [Shapiro, 1986] prevents one term from substituting for two different variables.

CassieW identifies the south and west walls by performing the act `assertWall(`*d*`,`*x*`)`, which causes her to believe that the column of cells just east of column *x* or the row of cells just south of row *x* are not rooms, as shown in Fig. 4.

12 Dead Reckoning

CassieW knows that she starts in `cell(0,0)` facing east, but she has to keep track of her position and facing afterwards by dead reckoning. She can keep track of the direction she's facing by her knowledge of the effects of turning:

```
all(x)({Isa(x,Number)}v=>{ActPlan(assertWall(east,x),
                            withsome/3(?i,Successor(x,?i),
                              withall/3(?n,Isa(?n,Number),believe(~Isa(cell(?i,?n),Room)))),
                          ActPlan(assertWall(south,x),
                            withsome/3(?i,Successor(x,?i),
                              withall/3(?n,Isa(?n,Number),believe(~Isa(cell(?n,?i),Room))))})
```

Figure 4: CassieW's plans for believing where the east and south walls are.

```
all(r)(In(r) => all(d)(Facing(d) => (all(r2)(Adjacent3(r,r2,d) =>
  Effect(go(forward), whendo(Isa(r2,Room),
                do-all({believe(In(r2)),
                        snif(if(~Have(gold), withsome(?r3, VisitedFrom(r2,?r3), do(nothing),
                                                believe(VisitedFrom(r2,r)))))}))))))))
```

Figure 5: CassieW's belief about the effects of going forward.

```
all(d,dl,dr)({Facing(d), Clockwise(dl,d),
                         Clockwise(d,dr)}
         &=> {Effect(go(right), Facing(dr)),
              Effect(go(left), Facing(dl))})
```

(The formula {P1,...,Pn} &=> {Q1,...,Qm} denotes the proposition that if all the Pi are true, then so are all the Qj.)

CassieW's belief about the effects of going forward is shown in Fig. 5. It has three parts: 1) after going forward, she is in the room in front of her; 2) she visited this new room from the previous room; 3) if she earlier visited this new room from some other room, just remember that occurrence. The accuracy of these effects relies on the fact that, if this act of going forward resulted in CassieW's feeling a bump, the effects of the bumping will be believed before these effects of going forward. Therefore, the cell in front of her will not be a room, and these effects will not be believed. The set of VisitedFrom(*r,r1*) beliefs (meaning "I visited room *r* from room *r1*") will form a trail of "crumbs" CassieW will follow after she finds the gold (see §15). Avoiding new visits in favor of old visits cuts loops in this trail.

Note that the construct withsome(?*x*, *p*(?*x*), do(nothing), *a*) is the autoepistemic policy, perform *a* if you know of no ?*x* for which you believe *p*(?*x*).

13 Finding the Gold

CassieW's strategy to find the gold is to explore the cave semi-randomly. Her explore(cave) plan uses a three-way categorization of the rooms: rooms she has already visited are "old rooms"; rooms she has not yet visited are "new rooms"; new rooms that she knows are safe are "safe new rooms." CassieW's rule for categorizing rooms is shown in Fig. 6. The proposition *RoomType*(r,r2,d) means that r2 is of the given *RoomType*, and is just d-ward of room r.

CassieW's semi-random exploration is further controlled by her level of boredom, which is represented by Bored(*i*), denoting the proposition "My level of boredom is *i*", for $0 \leq i \leq max_b$. Currently max_b is 4, so CassieW begins with the state constraint and-dor(1,1){Bored(0), Bored(1), Bored(2), Bored(3), Bored(4)} and the initial belief that Bored(0). She increases her level of boredom, up to max_b, with the act do(raiseBoredom):

```
ActPlan(do(raiseBoredom),
  snif(if(~Bored(4),
       withsome/3(?n1, Bored(?n1),
         withsome/3(?n2,Successor(?n1,?n2),
                    believe(Bored(?n2)))))))
```

CassieW's plan for exploring the cave is shown in Figure 7. If she can move to a safe new room, she'll do that, and set her boredom level to 0; if she can't find such a room, and she's not totally bored, she'll go to an old room (she needn't do(perceive) there), and increase her boredom level; if she can't find a safe new room, and she's totally bored, she'll move to any new room (even though she might die), and set her boredom level to 0; in any other case, she'll make a random move. After making one move, CassieW continues exploring.

CassieW makes a random move by going forward 50% of the time, and right or left 25% of the time each. However, moving randomly is boring:

```
ActPlan(do(random), snsequence(
          prdo-one({pract(50,move(forward)),
                    pract(25,go(right)),
                    pract(25,go(left))}),
          do(raiseBoredom))).
```

CassieW's plan for explore(cave) uses the act turn(*d*), where *d* is some direction. CassieW's plans for turning are shown in Fig. 8

14 Shooting the Wumpus

CassieW does not go to any particular effort to try to shoot the wumpus. She explores the cave, looking for the gold. If she happens to locate the wumpus before getting the gold, she adopts the policy that if she happens to be in a room in the same row or column as the wumpus, then, if she still has her arrow, she should turn toward the wumpus and shoot:

```
{~Have(gold), Alive(wumpus)} &=> {
   all(r,d)(WumpusAhead(r,d) =>
     whendo(In(r), snif(if(Have(arrow),
           snsequence(turn(d),do(shoot))))))}
```

WumpusAhead(*r,d*) denotes the proposition that the wumpus is somewhere *d*-ward of room *r*. It can be

```
all(r)(Isa(r,Room) => all(r2,d)({Adjacent3(r,r2,d), Isa(r2,Room)} &=> {
  (Visited(r2) => OldRoom(r,r2,d)),
  {~Visited(r2)} v=> {NewNextRoom(r,r2,d), Safe(r2) => SafeNewRoom(r,r2,d)}}))
```

Figure 6: CassieW's rule for categorizing rooms.

```
~Have(gold) => (all(r1)(In(r1) => ActPlan(explore(cave),
   snsequence(withsome({?r2,?d1}, SafeNewRoom(r1,?r2,?d1),
               snsequence3(turn(?d1), move(forward), believe(Bored(0))),
               snif({if(Bored(4), withsome({?r3,?d2}, NewNextRoom(r1,?r3,?d2),
                                  snsequence3(turn(?d2), move(forward), believe(Bored(0))),
                                  do(random))),
                     else(withsome({?r3,?d2}, OldRoom(r1,?r3,?d2),
                            snsequence3(turn(?d2), go(forward), do(raiseBoredom)),
                            do(random)))})),
            explore(cave)))))
```

Figure 7: CassieW's plan for exploring the cave.

inferred from propositions of the form `LocationAhead(r,d,r1)`, which denotes the proposition that room *r1* is somewhere *d*-ward of room *r*, and is the transitive closure of `Adjacent3`:

```
all(r,d,r1)(Adjacent3(r,r1,d)
             => LocationAhead(r,d,r1))
all(r,d,r1,r2)({LocationAhead(r,d,r1),
                 LocationAhead(r1,d,r2)}
              &=> {LocationAhead(r,d,r2)})
all(r1)(Contains(r1, wumpus) =>
        (all(d,r2)(LocationAhead(r2,d,r1)
                    => WumpusAhead(r2,d))))
```

CassieW initially believes that `Have(arrow)`. An effect of shooting is that she no longer has it: `Effect(do(shoot), ~Have(arrow))`.

15 Getting Home

As soon as CassieW believes she has the gold, she changes the `explore(cave)` plan to one of finding home: `Have(gold) => ActPlan(explore(cave), find(home))`. Her plan for finding home, shown in Fig. 9 is to go back the way she came, using her `VisitedFrom` beliefs as "crumbs" (see §11), picking them up as she goes, and stopping when she reaches `cell(0,0)`. She doesn't need to `do(perceive)` on her way home.

16 Results

We ran a series of trials with $max_x = max_y = 6$, and the width and height of the cave each independently 4 ± 1. We ran enough trials so that there were 10 cases of each of 3 types: P — it is possible for CassieW to get to the gold moving only into safe rooms; I — it is impossible for CassieW to get to the gold; PS — the gold is perceptually screened from CassieW, *i.e.*, she can get the gold only by taking a risky move. The average scores are shown in the table below.

Type	Nwon	AvgWon	Nlost	AvgLost	Avg
P	9	985.0	1	-2023.0	684.2
PS	2	982.0	8	-2006.5	-1408.8
I	0		10	-2008.5	-2008.5

17 Conclusions

The SNePS/SNeRE knowledge representation, reasoning and acting system provides an expressive language for building agents that perform integrated first-person, on-line reasoning and acting. GLAIR is an effective architecture for building such agents. CassieW, a wumpus world agent, is an excellent example of the use of GLAIR and SNePS/SNeRE.

References

[Alchourrón *et al.*, 1985] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The Journal of Symbolic Logic*, 50(2):510–530, June 1985.

[Chalupsky and Shapiro, 1996] Hans Chalupsky and Stuart C. Shapiro. Reasoning about incomplete agents. In *Proceedings of the Fifth International Conference on User Modeling (UM-96)*, pages 169–177. User Modeling, Inc., 1996.

[Hexmoor and Shapiro, 1997] Henry Hexmoor and Stuart C. Shapiro. Integrating skill and knowledge in expert agents. In P. J. Feltovich, K. M. Ford, and R. R. Hoffman, editors, *Expertise in Context*, pages 383–404. AAAI Press/MIT Press, Cambridge, MA, 1997.

[Hexmoor *et al.*, 1993] Henry Hexmoor, Johan Lammens, and Stuart C. Shapiro. Embodiment in GLAIR: a grounded layered architecture with integrated reasoning for autonomous agents. In Douglas D. Dankel II and John Stewman, editors, *Proceedings of The Sixth Florida AI Research Symposium (FLAIRS 93)*, pages 325–329. The Florida AI Research Society, April 1993.

[Ismail and Shapiro, 2000] Haythem O. Ismail and Stuart C. Shapiro. Two problems with reasoning and acting in time. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR 2000)*, pages 355–365, San Francisco, 2000. Morgan Kaufmann.

```
all(d)(Isa(d, Direction) => ActPlan(turn(d),
  snif(if(~Facing(d), withsome/3(?f, Facing(?f), snif({if(Clockwise(d,?f), go(left)),
                                                       if(Clockwise(?f,d), go(right)),
                                                       else(turn(around))}))))))
ActPlan(turn(around), snsequence(go(right), go(right)))
```

Figure 8: CassieW's plans for turning.

```
all(r)(In(r) => ActPlan(find(home), snif({if(In(cell(0,0)), do(stop)),
  else(snsequence(withsome/3(?r2,VisitedFrom(r, ?r2),
                    withsome/3(?d,Adjacent3(r,?r2,?d),
                      snsequence3(turn(?d), go(forward), disbelieve(VisitedFrom(r,?r2))))),
                  find(home)))})))
```

Figure 9: CassieW's plan for finding home.

[Ismail, 2001] Haythem O. Ismail. *Reasoning and Acting in Time*. Ph.D. dissertation, Technical Report 2001-11, University at Buffalo, The State University of New York, Buffalo, NY, August 2001.

[Kumar and Shapiro, 1994a] Deepak Kumar and Stuart C. Shapiro. Acting in service of inference (and *vice versa*). In Douglas D. Dankel II, editor, *Proceedings of The Seventh Florida AI Research Symposium (FLAIRS 94)*, pages 207–211. The Florida AI Research Society, May 1994.

[Kumar and Shapiro, 1994b] Deepak Kumar and Stuart C. Shapiro. The OK BDI architecture. *International Journal on Artificial Intelligence Tools*, 3(3):349–366, March 1994.

[Kumar, 1996] Deepak Kumar. The SNePS BDI architecture. *Decision Support Systems*, 16(1):3–19, January 1996.

[Martins and Shapiro, 1988] João P. Martins and Stuart C. Shapiro. A model for belief revision. *Artificial Intelligence*, 35:25–79, 1988.

[Russell and Norvig, 1995] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 1995.

[Shapiro and Ismail, 2003] Stuart C. Shapiro and Haythem O. Ismail. Anchoring in a grounded layered architecture with integrated reasoning. *Robotics and Autonomous Systems*, 43(2–3):97–108, May 2003.

[Shapiro and Rapaport, 1987] Stuart C. Shapiro and William J. Rapaport. SNePS considered as a fully intensional propositional semantic network. In Nick Cercone and Gordon McCalla, editors, *The Knowledge Frontier*, pages 263–315. Springer-Verlag, New York, 1987.

[Shapiro and Rapaport, 1991] Stuart C. Shapiro and William J. Rapaport. Models and minds: Knowledge representation for natural-language competence. In Robert Cummins and John Pollock, editors, *Philosophy and AI: Essays at the Interface*, pages 215–259. MIT Press, Cambridge, MA, 1991.

[Shapiro *et al.*, 2000] Stuart C. Shapiro, Haythem O. Ismail, and John F. Santore. Our dinner with Cassie. In *Working Notes for the AAAI 2000 Spring Symposium on Natural Dialogues with Practical Robotic Devices*, pages 57–61, Menlo Park, CA, 2000. AAAI.

[Shapiro *et al.*, 2004] Stuart C. Shapiro *et al. SNePS 2.6.1 User's Manual*. Department of Computer Science and Engineering, University at Buffalo, The State University of New York, Buffalo, NY, 2004.

[Shapiro, 1979] S. C. Shapiro. Numerical quantifiers and their use in reasoning with negative information. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 791–796. Morgan Kaufmann, San Mateo, CA, 1979.

[Shapiro, 1986] Stuart C. Shapiro. Symmetric relations, intensional individuals, and variable binding. *Proceedings of the IEEE*, 74(10):1354–1363, October 1986.

[Shapiro, 1989] Stuart C. Shapiro. The CASSIE projects: An approach to natural language competence. In J. P. Martins and E. M. Morgado, editors, *EPIA 89: 4th Portugese Conference on Artificial Intelligence Proceedings, Lecture Notes in Artificial Intelligence 390*, pages 362–380. Springer-Verlag, Berlin, 1989.

[Shapiro, 1991] Stuart C. Shapiro. Cables, paths and "subconscious" reasoning in propositional semantic networks. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 137–156. Morgan Kaufmann, Los Altos, CA, 1991.

[Shapiro, 1993] Stuart C. Shapiro. Belief spaces as sets of propositions. *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, 5(2&3):225–235, April–September 1993.

[Shapiro, 1998] Stuart C. Shapiro. Embodied Cassie. In *Cognitive Robotics: Papers from the 1998 AAAI Fall Symposium, Technical Report FS-98-02*, pages 136–143. AAAI Press, Menlo Park, California, October 1998.

[Shapiro, 2000] Stuart C. Shapiro. SNePS: A logic for natural language understanding and commonsense reasoning. In Łucja Iwańska and Stuart C. Shapiro, editors, *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*, pages 175–195. AAAI Press/The MIT Press, Menlo Park, CA, 2000.