```
/*--------------------------------------------------------------------------+
|                                                                            |
|  This program is a demonstration of Q-learning.  To the learning          |
|  system, its environment is a black box from which it has several         |
|  signal lines and a reinforcement line.  Its task is to learn to give     |
|  responses which maximize the scalar signals on its reinforcement line.   |
|                                                                            |
|  The system must must learn to correlate its state in the environment     |
|  with the future reinforcements it will see for each of its candidate     |
|  actions.  Therefore, the system must determine its state from the        |
|  signal values it gets from the environment.  For the current             |
|  demonstration, the system does no feature extraction;  instead, it       |
|  determines its state according to a built-in look-up table.  This        |
|  table is the "boxes" state representation of Barto, Sutton, and          |
|  Anderson, described in their paper, "Neuronlike Adaptive Elements That   |
|  Solve Difficult Learning Control Problems," IEEE Trans. Syst., Man,      |
|  Cybern., Vol. SMC-13, pp. 834-846, Sep.- Oct. 1983.                      |
|                                                                            |
|  The Q-value update is the following, if the system takes action a from  |
|  state s at time t, and arrives at state ss with feedback r at time t+1 : |
|                                                                            |
|  Q(t+1, s, a) = Q(t, s, a)                                                |
|               + \alpha (r + \gamma max_{b}Q(t,ss, b) - Q(t, s, a))        |
|                                                                            |
|  To apply this system to different problems:                              |
|     1. Change the number of input parameters                              |
|     2. Replace comparisons of Q(state, 0) and Q(state, 1) with expressions |
|          for choosing the maximum over i of Q(state, i)                   |
|     3. Rewrite the state identification routine.  Ideally, this should not |
|          be a set of pre-defined "boxes," but the controller should       |
|          do feature extraction on-line, as it learns.                     |
|                                                                            |
+--------------------------------------------------------------------------*/
/*
 $Log: q.c,v $
 * Revision 1.1.1.1  1995/02/10  21:49:24  finton
 * Corrected bug in setting predicted_value for failure state,
 * where cur_state = -1.  The predicted value should be 0.0.
 *
 * Revision 1.1  1994/11/17  19:49:17  finton
 * Initial revision
 *
 */
#include

#define sqr(x) ( (x) * (x) )
#define W_INIT 0.0
#define NUM_BOXES 162
extern int RND_SEED;
static float ALPHA = 0.5;             /* learning rate parameter */
static float BETA = 0.0;              /* magnitude of noise added to choice */
static float GAMMA = 0.999;           /* discount factor for future reinf */

static float q_val[NUM_BOXES][2];     /* state-action values */

static first_time = 1;
static int cur_action, prev_action;
static int cur_state, prev_state;

static char rcs_controller_id[] = "$Id: q.c,v 1.1.1.1 1995/02/10 21:49:24 finton Exp $";

/*--------------------------------------------------------------------------+
|   get_action : returns either 0 or 1 as action choice;                    |
|               accepts five "black box" inputs, the first four of which are |
|                  system variables, and the last is a reinforcement signal. |
|               Note that reinf is the result of the previous state and     |
|                  action.                                                   |
+--------------------------------------------------------------------------*/

int get_action(float x,           /* system variables == state information */
            float x_dot,
            float theta,
            float theta_dot,
            float reinf)       /* reinforcement signal */
{
   int i,j;
   float predicted_value;         /* max_{b} Q(t, ss, b) */
   double rnd(double, double);
   int get_box(float x, float x_dot, float theta, float theta_dot);  /*state*/
   void srandom(int);
   void reset_controller(void);    /* reset state/action before new trial */

   if (first_time) {
```

```
    first_time = 0;
    reset_controller();   /* set state and action to null values */

    for (i = 0; i < NUM_BOXES; i++)
      for (j = 0; j < 2; j++)
        q_val[i][j] = W_INIT;

    printf("Controller: %s\n", rcs_controller_id);
    printf("... setting learning parameter ALPHA to %.4f.\n", ALPHA);
    printf("... setting noise parameter BETA to %.4f.\n", BETA);
    printf("... setting discount parameter GAMMA to %.4f.\n", GAMMA);
    printf("... random RND_SEED is %d.\n", RND_SEED);
    srandom(RND_SEED);    /* initialize random number generator */
  }

  prev_state = cur_state;
  prev_action = cur_action;
  cur_state = get_box(x, x_dot, theta, theta_dot);

  if (prev_action != -1)  /* Update, but not for first action in trial */
  {
    if (cur_state == -1)
      /* failure state has Q-value of 0, since the value won't be updated */
      predicted_value = 0.0;
    else if (q_val[cur_state][0] <= q_val[cur_state][1])
       predicted_value = q_val[cur_state][1];
    else
       predicted_value = q_val[cur_state][0];

    q_val[prev_state][prev_action]
      += ALPHA * (reinf + GAMMA * predicted_value
                  - q_val[prev_state][prev_action]);
  }

  /* Now determine best action */
  if (q_val[cur_state][0] + rnd(-BETA, BETA) <= q_val[cur_state][1])
    cur_action = 1;
  else
    cur_action = 0;

  return cur_action;
}


double rnd(double low_bound, double hi_bound)
/* rnd scales the output of the system random function to the
 * range [low_bound, hi_bound].
 */
{
  long random(void);         /* system random number generator */

  double highest = (double) RAND_MAX;
  /* if RAND_MAX is not defined, try ((1 << 31) -1) */
  return (random() / highest) * (hi_bound - low_bound) + low_bound;
}


void reset_controller(void)
{
  cur_state = prev_state = 0;
  cur_action = prev_action = -1;   /* "null" action value */
}


/* The following routine was written by Rich Sutton and Chuck Anderson,
    with translation from FORTRAN to C by Claude Sammut  */

/*----------------------------------------------------------------------
  get_box:  Given the current state, returns a number from 0 to 161
 designating the region of the state space encompassing the current state.
 Returns a value of -1 if a failure state is encountered.
----------------------------------------------------------------------*/

#define one_degree 0.0174532   /* 2pi/360 */
#define six_degrees 0.1047192
#define twelve_degrees 0.2094384
#define fifty_degrees 0.87266

int get_box(float x, float x_dot, float theta, float theta_dot)
{
 int box=0;

 if (x < -2.4 ||
```

```
    x > 2.4  ||
    theta < -twelve_degrees ||
    theta > twelve_degrees)        return(-1); /* to signal failure */

  if (x < -0.8)                box = 0;
  else if (x < 0.8)            box = 1;
  else                         box = 2;

  if (x_dot < -0.5)                ;
  else if (x_dot < 0.5)            box += 3;
  else                             box += 6;

  if (theta < -six_degrees)            ;
  else if (theta < -one_degree)        box += 9;
  else if (theta < 0)                  box += 18;
  else if (theta < one_degree)         box += 27;
  else if (theta < six_degrees)        box += 36;
  else                                 box += 45;

  if (theta_dot < -fifty_degrees)     ;
  else if (theta_dot < fifty_degrees)  box += 54;
  else                                 box += 108;

  return(box);
}
```