Appendix A

COMPUTER CODE FOR THE SIMPLE COMBAT MODEL

The code presented here is for a single battle in Fig. 2 with $B_0 = 500$ and $R_0 = 1000$. It represents the fundamental unit of our investigations. All other computations used code such as this for the calculation of the outcome of an individual battle.

```
/* delayrb_r: Lanchester model with reinforcement and withdrawal
 *
 * Dewar and Gillogly:  1988-1990.
 */

char *hdr = "$Id$";

#include <stdio.h>
#include <math.h>

typedef double real;

real RI = 1000;                 /* Initial number of Red troops */
real BI = 500;                  /* Initial number of Blue troops */

real c1 = 1. / 2048.;           /* Defender's attrition rate */
real c2 = 1. / 512.;            /* Attacker's   "        "   */

real rBA = 4;                   /* Blue calls for reinforcements at 1:4 */
real rRA = 2.5;                 /* Red calls for reinforcements at 2.5:1 */

real rBW = 10;                  /* Blue withdraws at 1:10 */
real rRW = 1.5;                 /* Red withdraws at 1.5:1 */

real rBAA = .80;                /* Blue calls for more if attrition is high*/
real rRAA = .80;                /* Red  " */

real rBAW = .70;                /* Blue withdraws when down to this */
real rRAW = .70;                /* Red    "       */

int B_delay = 70;               /* Arrival time delay of reinforcements */
int R_delay = 70;               /* Arrival time delay of reinforcements */

int B_maxchunks = 5;            /* How many chunks may they use? */
int R_maxchunks = 5;            /* How many chunks may they use? */
int b_tot_reinf = 1500;         /* Total reinforcements */
int r_tot_reinf = 1500;

real a1;                        /* Reinforcement chunk size for Blue */
real a2;                        /*       "         "    "    "  Red */

int bchunks, rchunks;           /* How many chunks have Blue and Red ordered? */

real B_reinf, R_reinf;          /* Total reinforcements tossed in */

int B_step, R_step;             /* Arrival time step of reinforcements */
int B_ordered, R_ordered;       /* How many reinforcements ordered? */

real B_with_thresh, R_with_thresh, B_reinf_thresh, R_reinf_thresh;

real rc;                        /* Force ratio: Red/Blue */
real R, B;                      /* Current number of Red/Blue troops */

#define YES 1
#define NO 0
#define STOP 1
#define GO 0

#define MAXITER 200000


main()
{
        long i;

        B = BI;
        R = RI;

        B_reinf = R_reinf = 0;        /* Total reinforcements so far */
        B_ordered = R_ordered = 0;

        B_with_thresh = BI * rBAW;    /* Thresholds */
        B_reinf_thresh = BI * rBAA;

        R_with_thresh = RI * rRAW;
        R_reinf_thresh = RI * rRAA;

        B_step = R_step = 0;
        bchunks = rchunks = 0;

        a1 = (real) b_tot_reinf / B_maxchunks; /* Size of Blue blocks */
        a2 = (real) r_tot_reinf / R_maxchunks; /* Size of Red blocks */
        for (i = 0; i < MAXITER; i++)  /* ~ 1/2-hour intervals */
        {
            if (B <= 0) B = .0001;  /* Don't divide by 0 */
            rc = R/B;               /* Force ratio */

            call_reinforcements(i); /* Call for reinforcements */
            reinforce(i);           /* Do the reinforcement */
            if (withdraw(i) == STOP) break; /* See if one side quits */
            attrit();               /* Blue and Red attrition */

        }
        exit(0);
}

call_reinforcements(i)  /* see if we need to call for reinforcements */
int i;                  /* Current time */
{
        /* Blue reinforcement based on attrition */
        if (B_ordered == 0 && bchunks < B_maxchunks &&
            B < B_reinf_thresh)
        {
              B_step = i + B_delay;
              B_ordered = a1;
              bchunks++;
              printf("Blue calls for reinforcements/A  at %d.\n", i);
        }
        /* Blue reinforcement based on force ratio */
        if (B_ordered == 0 && rc > rBA && bchunks < B_maxchunks)
        {
              B_step = i + B_delay;
              B_ordered = a1;
              bchunks++;
              printf("Blue calls for reinforcements/FR at %d.\n", i);
        }
        /* Red reinforcement based on attrition */
        if (R_ordered == 0 && rchunks < R_maxchunks &&
            R < R_reinf_thresh)
        {
              R_step = i + R_delay;
              R_ordered = a2;
              rchunks++;
              printf("Red  calls for reinforcements/A  at %d.\n", i);
        }
        /* Red reinforcement based on force ratio */
        if (R_ordered == 0 && rc < rRA && rchunks < R_maxchunks)
        {
              R_step = i + R_delay;
            R_ordered = a2;
            rchunks++;
            printf("Red  calls for reinforcements/FR at %d.\n", i);
      }
}


reinforce(i)    /* Do the reinforcement */
int i;
{
            if (B_ordered > 0 && B_step <= i)  /* B reinf arrive */
            {     B += B_ordered;
                  B_reinf += B_ordered;
                  B_ordered = 0;
                  printf("Blue reinforcements arrive at %d.\n", i);
            }
            if (R_ordered > 0 && R_step <= i)   /* R reinf arrive */
            {     R += R_ordered;
                  R_reinf += R_ordered;
                  R_ordered = 0;
                  printf("Red  reinforcements arrive at %d.\n", i);
            }

}


withdraw(i)     /* See if one side wants to withdraw */
int i;
{
      /* Check for withdrawal based on force ratio or attr */

      if (B < B_with_thresh || rc > rBW)
      {
            printf("Blue withdraws at %d.\n", i);
            return STOP;
      }
      if (R < R_with_thresh || rc < rRW)
      {
            printf("Red  withdraws at %d.\n", i);
            return STOP;
      }

      return GO;
}
```

```
attrit()        /* Throw stones at frogs in sport */
{
      real b, t;

      b = B;                     /* B and R troop strength */
      t = c1 * R;                /* Blue attrition */
      B -= t;
      t = c2 * b;                /* frogs die in earnest */
      R -= t;
}
```