Life—Multiplayer

John Conway's Game of Life is a sort of mathematical game where simple rules can lead to complex patterns. In its original incarnation life is designed to set up and played on a giant checkerboard by hand but it proved to be a perfect thing for a computer to do. The rules are:

- A token with 2 or 3 neighbors survives.
- A token with more or less than 2 or 3 neighbors dies from loneliness or overpopulation.
- An empty space with exactly 3 neighbors gives birth to a new token.

These rules are enough for the "zero player" game. However, to make it a multiplayer game requires a few additional rules:

- On every turn each player gets to place a piece on the board.
- When giving birth the player with the majority of the 3 neighbors of an empty cell gets the new cell.
- If no one has the majority (ie a 3 way tie) the cell becomes a hybred that no one controls.
- In every turn there is an advantage to turn order. In order to negate that turns are given in round robin order with a different player going first each time.

You can either play a small game on a 5x5 board (good for 2 player games) or a large game on 25x25

* lifemultiplayer.c linsting begins: *\

```
/* Cell Life ver 2008Dec05
 * by Joseph Larson
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define F(a,b) for (a = 0; a < (b); a++)
#define DIG v * tp(5) + w * tp(4) + x * tp(3) + y * 100 + z * 10
#define TURN ((c + f + g) % players)
#define MAX_W 26
#define MAX_H 19
#define PIECES 3

int bd[MAX_W][MAX_H], dis[8][2], pop[5], surv[5][55], hyb[10];
int height, width, players;
char tokens[6] = ".OX*#@";

void intro (void) {
  printf ("Cell Life\n---------\n\n"
  "The rules of this game are based on Conway's Game of Life. Tokens are\n"
  "placed on the board, then with every generation a token will live, die, or\n"
  "be born based on these rules :"
  " o A token with 2 or 3 neighbors survives.\n"
  " o A token with more or less than 2 or 3 neighbors dies from lonelyness or\n"
  "   overpopulation.\n"
  " o An empty space with exactly 3 neighbors gives birth to a new token.\n\n"
  "When giving birth the player with the majority of the 3 neighbors of an\n"
  "empty cell gets the new cell. If no one has the majority (in the case of\n"
  "a 3 way tie) the cell becomes a hybred that no one controls.\n"
  "Remember to count the number of neighbors same as in minesweeper. In other\n"
  "words a cell can have up to 8 neighbors.\n\n"
  "The first three moves (Generation 0) are when you place your initial\n"
  "pieces.\n\n"
  "Remember to pay close attention to who's turn it is as every turn a\n"
  "different player gets to go first.\n\n"
  "Good Luck.\n\n");
}

int tp (int x) {return ((x == 0) ? 1 : 10 * tp (x - 1)); }

void init (void) {
  int p, v, w, x, y, z, i, sum;

  srand (time (NULL));
  F(p, 5) {
    i  = 0;
    F(v, 4) F(w, 4) F(x, 4) F(y, 4) F(z, 4) {
      sum = v + w + x + y + z;
      if (sum == 3 || sum == 2) surv[p][i++] = (p + 1) + DIG;
      else if (sum == 1)
        surv[p][i++] = DIG + 2 * tp(p + 1);
    }
  }
  i = 0;
  F(v, 2) F(w, 2) F(x, 2) F(y, 2) F(z, 2) {
    sum = v + w + x + y + z;
    if (sum == 3) hyb[i++] = DIG;
  }
  i = 0;
```

Listing continued on next page...

* Listing continued from previous page *\

```
  F (x, 3) F (y, 3) if (!(x == y && y == 1)) {
    dis[i][0] = x - 1; dis[i++][1] = y - 1;
  }
}

void print_bd (void) {
  int x, y;

  printf ("  "); /* 2 spaces */
  F (x, width) putchar (" 12"[x / 10]);
  putchar ('\n');
  F (y, (height + 2)) {
    F (x, (width + 2)) {
      if (!y || y > height) {
        if (!x || x > width) {
          printf ("  "); /* 2 spaces */
        } else putchar ('0' + (x % 10));
      } else if (!x) {
        printf ("%2d", y);
      } else if (x > width) {
        printf ("%-2d", y);
      } else putchar (tokens [bd[x][y] % 10]);
    }
    switch (y) {
      case 0 : printf ("\tKey:"); break;
      case 1 : printf ("\t----"); break;
      case 2 :
      case 3 :
      case 4 :
      case 5 :
        printf ("\t%c - Player %d", tokens[y - 1], y - 1);
        if (!pop[y - 2]) printf (" DEAD");
        break;
      case 6 : printf ("\t%c - Hybred", tokens[5]); break;
    }
    printf ("\n");
  }
  printf ("  "); /* 2 spaces */
  F (x, width) putchar (" 1234"[x / 10]); putchar ('\n');
}

int survives (int x) {
  int k, p;

  F (p, 5) F(k, 55) if (surv[p][k] == x) return p + 1;
  F (k, 10) if (hyb[k] == x) return 5;
  return 0;
}

void generation (void) {
  int x, y, k, p;

  F (x, width + 1) F (y, height + 1) if (p = (bd[x][y] % 10))
    F (k, 8) bd[x + dis[k][0]][y + dis[k][1]] += tp (p);
  F (k, 5) pop[k] = 0;
  F (x, width + 2) F (y, height + 2)
    if (!y || y > height || !x || x > width) bd[x][y] = 0;
    else {
      bd[x][y] = survives (bd[x][y]);
      if (bd[x][y]) pop[bd[x][y] - 1]++;
    }
}
```

* Listing continued on next page… *\

board. In order to win it helps to be familiar with the pattens of the original game version of the game of life. Learn some stable patterns to hold in reserve and growing patterns to invade your opponents space. Since you can only add pieces to the board overpopulation is the best way to remove your opponents pieces from the board.

LifeMultiplayer was written by Joe Larson based on a 2 player BASIC game by Bryan Wyvill as found in 'BASIC Computer Games' edited by David H Ahl (c) 1978, which game was derived from John Conway's Game of Life as found in Scientific America, October 1970.

* Listing continued from previous page *\

```
void getxy (int *x, int *y) {
  printf ("<X,Y>: ");
  scanf ("%d %*c %d", x, y);
  while (*x > width || *x < 1 || *y > height || *y < 1) {
    printf ("Invalad location. Retry. x,y : ");
    scanf ("%d %*c %d", x, y);
  }
}

void compyturn (int p, int g) {
  int x, y, c, d, bestx, besty, r, bestr, k;
  int bdc[MAX_W][MAX_H], popc[5];

  if (g <= PIECES) if (g == 1) {
    bestx = rand() % (width - 1) + 1;
    besty = rand() % (width - 1) + 1;
  } else {
    F (x, width) F (y, width) if (bd[x][y] == (p + 1)) {bestx = x; besty = y;}
    r = 0;
    do {
      k = rand () % 8;
      if (bd[bestx + dis[k][0]][besty + dis[k][1]]) k = -1;
      else if ((bestx + dis[k][0]) > width || (bestx + dis[k][0]) < 1) k = -1;
      else if ((besty + dis[k][1]) > height || (besty + dis[k][1]) < 1) k = -1;
       r |= 1 << k;
    } while (k < 0 && r < 255);
    if (k < 0) {
      bestx = rand() % (width - 1) + 1; besty = rand() % (width - 1) + 1;
    } else { bestx += dis[k][0]; besty += dis[k][1]; }
  } else {
    F (c, height + 1) F (d, width + 1) bdc[c][d] = bd[c][d];
    F (k, 5) popc[k] = pop[k];
    bestx = rand() % (width - 1) + 1;
    besty = rand() % (width - 1) + 1;
    bestr = -99;
    F (y, height - 1) F (x, width - 1) if (!bd[x + 1][y + 1]) {
      bd[x + 1][y + 1] = p + 1;
      generation ();
      r = 0;
      F (k, 5) if (k == p) r += pop[k]; else r -= pop[k];
      if (r > bestr) {bestr = r; bestx = x + 1; besty = y + 1;}
      F (c, height + 1) F (d, width + 1) bd[c][d] = bdc[c][d];
      F (k, 5) pop[k] = popc[k];
    }
  }
  bd[bestx][besty] = p + 1;
  printf ("Computer Player %d (%c) : %d, %d\n", p + 1, tokens[p + 1], bestx,
besty);
}

int winner (void) {
  int c, w = 0;

  F (c, players) if (pop[c]) {if (w) return 0; else w = c + 1;}
  return (w) ? w : 6;
}

int getint (int low, int high) {
  int t;

  scanf ("%d", &t);
  while (t < low || t > high) {
    printf ("Invalad. Choose a number between %d and %d : ", low, high);
```

* Listing continued on next page… *\

* Listing continued from previous page *\

```
    scanf ("%d", &t);
  }
  return t;
}

void play (void) {
  int c, d, f, x, y, comp[4], g = 0;

  printf ("How many players? (2 - 4) ");
  players = getint (2, 4);
  F(c, players) {
    printf ("Player %d, (1) human or (2) computer? : ", c + 1);
    comp[c] = getint (1, 2) - 1;
  }
  printf ("(1) Small or (2) large game? ");
  c = getint(1, 2);
  width = (--c) ? MAX_W - 1 : 5;
  height = (c) ? MAX_H - 1 : 5;
  f = rand() % players;
  printf ("Each player gets %d pieces to play.\n", PIECES);

  F (c, players) pop[c] = 1;
  do {
    g++;
    F (c, players) {
      if (pop[TURN]) {
        print_bd ();
        if (!comp[TURN]) {
          printf ("Generation %d, player %d (%c) ", (g < PIECES) ? 0 : g - PIECES
            , TURN + 1, tokens[TURN + 1]);
          getxy (&x, &y);
          if (bd[x][y]) printf ("Occupied. Turn discarded.\n");
          else bd[x][y] = TURN + 1;
        } else compyturn (TURN, g);
      }
    }
    if (g >= PIECES) {
      print_bd ();
      printf ("\nPress ENTER to advance the generation:\n");
      getchar (); getchar ();
      generation ();
    }
  } while ((g < 0) || !(d = winner ()));

  print_bd();
  printf ((d == 6) ? "\n\nTotal extinction." :
    (d == 5) ? "\n\nHybred takeover!" : "\n\nPlayer %d wins!", d);
}

int main (void){

  intro ();
  init ();
  play ();
  getchar ();
  exit (0);
}
```