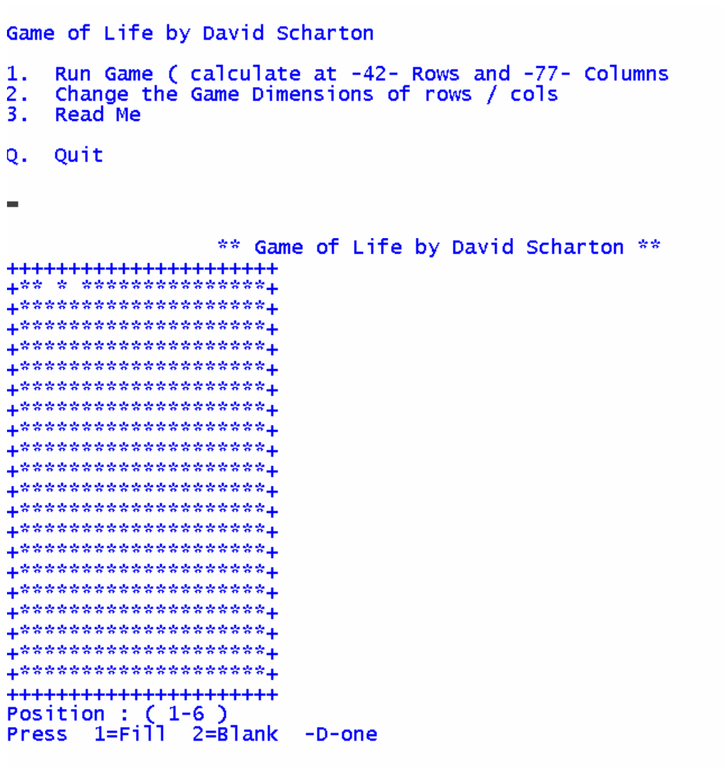
Game of Life – written in C++



```
Game of Life by David Scharton

1.  Run Game ( calculate at -42- Rows and -77- Columns
2.  Change the Game Dimensions of rows / cols
3.  Read Me

Q.  Quit

_

                 ** Game of Life by David Scharton **
++++++++++++++++++++++
+** * ***************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
+********************+
++++++++++++++++++++++
Position : ( 1-6 )
Press  1=Fill  2=Blank  -D-one
```

```
/*
David A. Scharton
CSIS 113a
C++ Programming
T-Th 8am-11am

Final

*****************************************************************************
*                          G A M E   O F   L I F E                          *
*****************************************************************************

GAME OF LIFE

The mathematician John Horton Conway invented the "Game of Life."  Though
not a "game" in any traditional sense, it provides interesting behavior that
is specified with only a few rules.  This project asks you to write a program
that allows you to specify an initial configuration.  The program follows
the rules of Life (listed shortly) to show the continuing behavior of the
configuration.

LIFE is an organism that lives in a discrete, two-dimensional world.  While this
world is actually unlimited, we don't have that luxury, so we restrict the array
to 80 characters wide by 22 character positions high.  If you have access to a
larger screen, by all means use it.

This world is an array with each cell capable of holding one LIFE cell.
```

```
Generations mark the passing of time.  Each generation brings births and deaths
to the LIFE community.  The births and deaths follow this set of rules:

   1.    We define each cell to have eight neighbor cells.  The neighbors of a
         cell are the cells directly above, below, to the right, to the left,
         diagonally above to the right and left, and diagonally below, to the
         right and left.

   2.    If an occupied cell has zero or one neighbor, it dies of loneliness. If
         an occupied cell has more than three neighbors, it dies of overcrowding.

   3.    If an empty cell has exactly three occupied neighbor cells, there is a
         birth of a new cell to replace the empty cell.

   4.    Births and deaths are instantaneous and occur at the changes of
         generation.  A cell dying for whatever reason may help cause birth, but a
         newborn cell cannot resurrect a cell that is dying, nor will a cell's
         death prevent the death of another , say, by reducing the local
         population.

Examples:    ***  becomes  *  then becomes *** again, and so on.

Notes:   Some configurations grow from relatively small starting configurations.
         Others move across the region.  It is recommended that for text output
         you use a rectangular char array with 80 columns and 22 rows to store the
         LIFE world's successive generations.  Use an * to indicate a living cell
         and use a blank to indicate an empty (or dead) cell.  If you have a
         screen with more rows than that, by all means make use of the whole
         screen.

         Suggestions: look for stable configurations.  That is, look for
         communities that repeat patterns continually.  The number of
         configuration in the repetitions is called the period. There are
         configurations that are fixed, that is, that continue without change.  A
         possible project is to find such configurations.

Hints:   Define a void function named generation that takes the array we call
         world, and 80-column by 22-row array of type char, which contains the
         initial configuration.
         The function scans the array and modifies the cells, marking the cells
         with births and deaths in accord with the rules listed previously.  This
         involves examining each cell in turn and either killing the cell, letting
         it lives, or if the cell is empty, deciding whether a cell should be
         born.  There should be a function display that accepts the array world
         and displays the array on the screen.  Some sort the time delay is
         appropriate between calls to generation and display.  To do this, your
         program should generate and display the next generation when you press
         Return.

         You are a liberty to automate this,  but automation is not necessary for
         the program.
*/


/*
David A. Scharton
```

```
CSIS 113a
C++ Programming
T-Th 8am-11am

Final

****************************************************************************
*                          G A M E   O F   L I F E                         *
****************************************************************************

*/

#include <iostream.h>
#include <cstdlib>

#include "ProjInfo.h"

const int xRow=42+2;
const int xCol=77+2;

int Xrow=xRow;
int Xcol=xCol;
int Xnum;

void dspCont();
void dspMenu();
void dspGMenu();
void subMenu();
void runGame(int num[][xCol], int &xrow, int &xcol, char fld);
void scrPaint(int num[][xCol],char fld);
void clcNeigh(int num[][xCol],int &xrow, int &xcol, char fld);
void chkTest(char fld[]);

int main()
{
      char menu1=' ';

      while(menu1 != 'Q' || menu1 != 'q' )
      {

            dspMenu();

            cout << "Q.  Quit" << endl;
            cout << ' ' << endl;

            cin >> menu1 ;

            switch(menu1)
            {
            case '1':

                        dspMenu();
                        subMenu();

                        break;

                  case '2':
                        int Erow, Ecol;

                        dspMenu();

                        cout << "    ** Row/Col Limits : " << xRow-2 << "/" <<
xCol-2 << " **\n " << endl;
                        cout << "       Enter Row & Col  "  ;
                        cin >> Erow >> Ecol;

                        if (Erow > 0 && Erow < (xRow+1)-2 )
                           if ( Ecol > 0 && Ecol < (xCol+1)-2 )
                           {
                                    Xcol = Ecol+2;
                                    Xrow = Erow+2;
                           }

                        break;

                  case '3':

                        system("cls");

                        cout << '\n' << endl;
                        cout << "                                The Game of Life
\n" << endl;
                        cout << "Generations mark the passing of time.  Each
generation brings births and" << endl;
                        cout << "deaths to the LIFE community.  The births and
deaths follow this set of rules:\n" << endl;

                        cout << "1. We define each cell to have eight neighbor
cells.  The neighbors of a cell" << endl;
                        cout << "   are the cells directly above, below, to the
right, to the left, diagonally" << endl;
                        cout << "   above to the right and left, and diagonally
below, to the right and left.\n" << endl;

                        cout << "2. If an occupied cell has zero or one neighbor,
it dies of loneliness. If an" << endl;
                        cout << "   occupied cell has more than three neighbors,
it dies of overcrowding.\n" << endl;

                        cout << "3. If an empty cell has exactly three occupied
neighbor cells, there is a birth" << endl;
                        cout << "   of a new cell to replace the empty cell. \n"
<< endl;

                        cout << "4. Births and deaths are instantaneous and occur
at the changes of generation." << endl;
                        cout << "   A cell dying for whatever reason may help
cause birth, but a newborn cell" << endl;
                        cout << "   cannot resurrect a cell that is dying, nor
will a cell's death prevent the" << endl;
                        cout << "   death of another , say, by reducing the local
population. " << endl;
```

```
                        cout << "\n" << endl;

                        dspCont();

                        system("cls");

                        cout << '\n' << endl;

                        cout << "                                  Assumptions
\n\n" << endl;

                        cout << "1. The complete matrix of cells are loaded with
lives, populated with *. \n" << endl;

                        cout << "2. If a cell is on the boarder of the matrix (
First or Last Row, or the First " << endl;
                        cout << "   or Last Column ), the cell treates the boarder
cells as died lives . . ." << endl;
                        cout << "   ex.  0,0  0,1 ...  1,0  2,0 ... etc. \n" <<
endl;

                        cout << "3. The not shell of the logic is if the cell has
3 lives around it.  It" << endl;
                        cout << "   has rebirth if it needs life and otherwises
DIES from crowding or loneliness.\n" << endl;

                        cout << "4. If it has 2 lives it stays the same. \n" <<
endl;
                        cout << "\n\n\n\n\n" << endl;

                        dspCont();

                  case 'Q':

                        break;
            }

            if(menu1 == 'Q' || menu1 == 'q' )
                  break;

      }

      system("cls");
      cout << "\n" << endl;
      cout << "ALL DONE ! \n \n " << endl;

      return 0;
}

void subMenu()
{
      char menu2=' ';
      int cnt[xRow][xCol];
      int pRow=1;
      int pCol=1;

      scrPaint(cnt,'C');
```

```
bool mnuflg=false;

while(menu2 != 'Q' || menu2 != 'q' )
{

      if(menu2=='d' || menu2=='D')
      {
            mnuflg=false;
      }

      if(mnuflg)
      {
            cout << "Position : ( " << pRow << "-" << pCol << " )" << endl;
            cout << "Press  1=Fill  2=Blank  -D-one  " << endl;
      }
      else
      {
            cout << "" << endl;
            cout << "Press -F-ill -C-lear -S-elect -R-un -Q-uit  " << endl;
      }

      cout << ' ' << endl;

      cin >> menu2 ;

      switch(menu2)
      {
            case 'D': case 'd':
                  scrPaint(cnt, 'P');
                  break;
            case 'S': case 's': case '1': case '2':
                  mnuflg=true;
                  runGame(cnt,pRow,pCol,menu2);
                  scrPaint(cnt, 'P');
                  break;

            case 'R': case 'r':
                  if(!mnuflg)
                  {
                        pRow=1,pCol=1;
                        runGame(cnt,pRow,pCol,menu2);
                        scrPaint(cnt, 'P');
                  }
                  break;

            case 'C': case 'c':
                  if(!mnuflg)
                  {
                        scrPaint(cnt, 'C');
                        pRow=1,pCol=1;
                  }
                  break;

            case 'F': case 'f':
                  if(!mnuflg)
                  {
                        scrPaint(cnt, 'F');
```

```
                              pRow=1,pCol=1;
                        }

                        break;

                  case 'Q':

                        break;
            }
            if(menu2 == 'Q' || menu2 == 'q' )
                  break;
      }
}

void dspMenu()
{
      system("cls");

      cout << ' ' << endl;
      cout << "Game of Life by David Scharton \n " << endl;
      cout << "1.  Run Game ( calculate at -" << Xrow-2 << "- Rows and -" << Xcol-
2 << "- Columns " << endl;
      cout << "2.  Change the Game Dimensions of rows / cols " << endl;
      cout << "3.  Read Me" << endl;
      cout << "  " << endl;

}

void dspGMenu()
{
      system("cls");
      cout << "                 ** Game of Life by David Scharton ** " << endl;

}

void dspCont()
{
      char chcc;
      cout << " \n Press C and then <RET> to Continue " ;
      cin >> chcc;
}

void scrPaint(int num[][xCol], char fld)
{
      dspGMenu();
      char line[80];

      for(int i=0;i<Xrow;i++)
      {
            for(int ii=0;ii<Xcol;ii++)
            {
                  if(fld=='F' || fld=='C')
                  {
                        if(i!= 0 && i!=Xrow-1 && ii!=0 && ii!=Xcol-1)
                              clcNeigh(num,i,ii,fld);
                        else
                              num[i][ii]='+';
                  }
                  line[ii]=num[i][ii];
                  cout << line[ii];
            }
            cout << '\n';
      }
}

void runGame(int num[][xCol],int &xrow, int &xcol, char fld)
{
      bool flgx=false;
      char line[80];

      if(xrow<Xrow && xcol<Xcol)
      {
            for(int i=0;i<Xrow;i++)
            {
                  for(int ii=0;ii<Xcol;ii++)
                  {
                        if(i!= 0 && i!=Xrow-1 && ii!=0 && ii!=Xcol-1)
                        {
                              switch(fld)
                              {
                              case 'S': case 's': case '1': case '2':
                                    if(ii==xcol && i==xrow && !flgx)
                                    {
                                          clcNeigh(num,xrow,xcol,fld);
                                          flgx=true;
                                    }

                                    break;
                              case 'R': case 'r':
                                    clcNeigh(num,i,ii,fld);

                                    break;
                              }
                        }
                  }
            }
            switch(fld)
            {
            case 'S': case 's': case '1': case '2':
                  xcol=xcol+1;
                  if(xcol>Xcol-2)
                  {
                        xcol=1;
                        if(xrow<Xrow-2)
                        {
                              xrow=xrow+1;
                        }
                        else
                              xrow=1;
                  }
            }
      }
}
```

```
void clcNeigh(int num[][xCol],int &xrow, int &xcol, char fld)
{
      switch(fld)
      {
      case 'C': case 'c': case '2':

            num[xrow][xcol]=' ';
            break;

      case 'F': case 'f': case '1':

            num[xrow][xcol]='*';                      // Populate the Cell
Initialization
            break;

      case 'R': case 'r':

            char srchArray[8];
            srchArray[0]=num[xrow-1][xcol-1];
            srchArray[1]=num[xrow][xcol-1];
            srchArray[2]=num[xrow+1][xcol-1];
            srchArray[3]=num[xrow-1][xcol];
            srchArray[4]=num[xrow+1][xcol];
            srchArray[5]=num[xrow-1][xcol+1];
            srchArray[6]=num[xrow][xcol+1];
            srchArray[7]=num[xrow+1][xcol+1];

            int xHit=0;

            for(int x=0;x<8;x++)
            {
                  if(srchArray[x]=='*')
                  {
                        xHit=xHit+1;
                  }

                  // chkTest("");
            }

            switch(xHit)
            {
            case 3:
                  num[xrow][xcol]='*';
                  break;
            case 2:
                  // Nothing
                  break;
            default:
                  // Die from loneliness or overcrowded
                   num[xrow][xcol]=' ';
                   break;
            }

            break;
      }
}
```

```
void chkTest(char fld[])
{
      char chkTest;

      cout << fld << endl;
      cin >> chkTest;

}
```