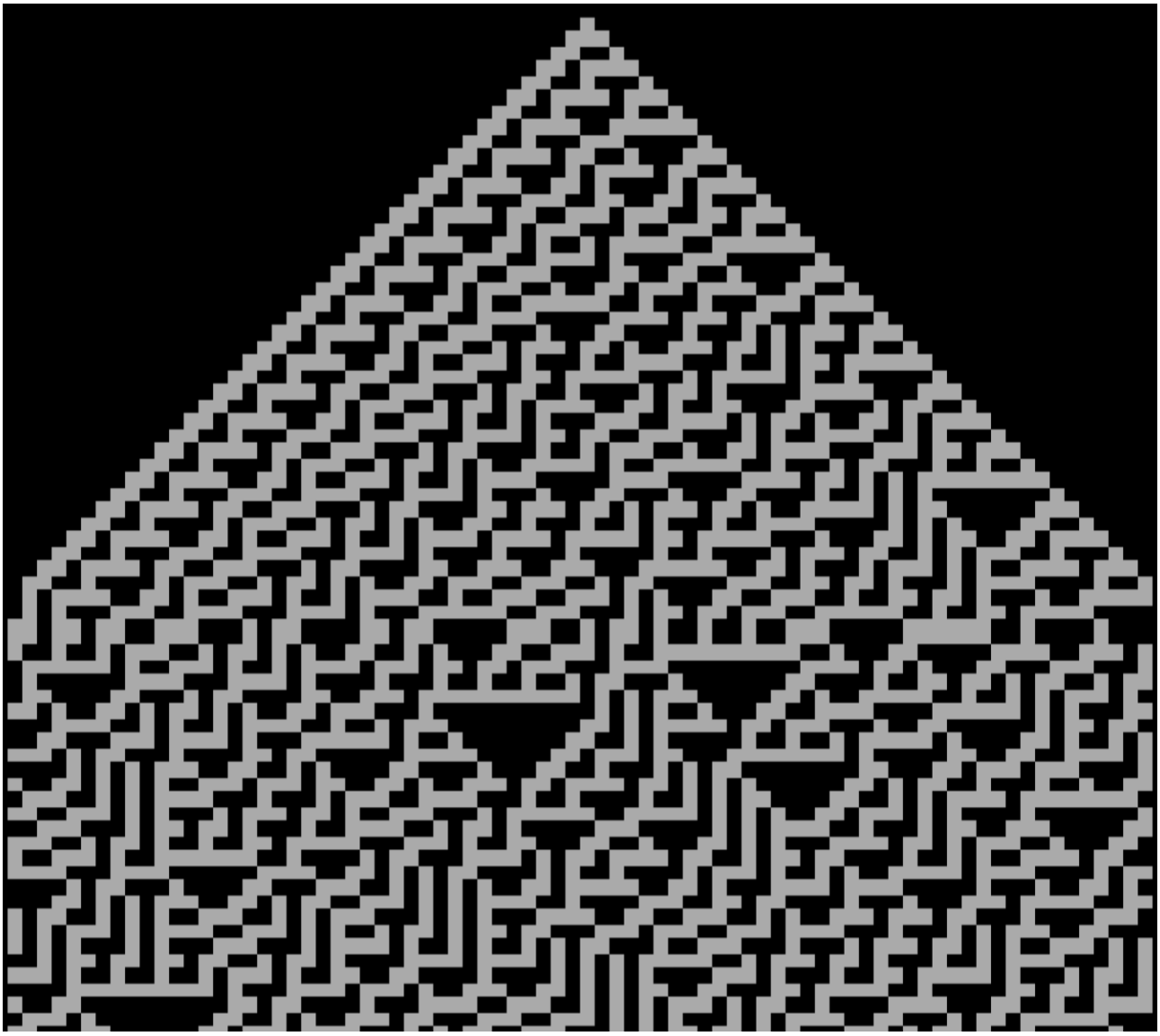
SIMULATING CELLULAR AUTOMATA

~ USING RULES OF CONWAY AND WOLFRAM ~

Harsh Vardhan Singh | Ishvik Kumar Singh | Utsav Munendra

Class: XI - A



Subject Teacher: Ms. Pinkie Srivastava

C++ Project File

ACKNOWLEDGEMENT

We would like to thank our computer teacher, Ms. Pinkie Srivastava for giving us the opportunity to make this project which allowed us to explore the world of cellular automata and also for teaching us the C++ programming language through which we were able to piece together our thought and ideas to form this program.

SPECIFICATIONS

The program is found to run successfully on the following hardware and software:

System	
Manufacturer	Dell
Model	Inspiron 3647 Desktop Computer
Processor	Intel Core i3-4125
RAM	4.00 GB
Operating System	Windows 10 Home Single Language
Type	x64-bit OS and Processor

Compiler	
IDE	Turbo C++ v3.2
Emulator	DOSBox v0.74

PURPOSE OF PROJECT

The purpose of this project is to stimulate life, with its complex and intelligent behavior. We know about life in the real world, governed by complex rules of sciences and are also familiar with familiar with life simulations as in sophisticated video games and virtual reality. In the words of David Shiffman, "A complex system is a system of elements, operating in parallel, with short-range relationships that as a whole exhibit emergent behavior." Simulating complex worlds with complex rules is easily conceivable, and it also highly practiced, which makes it worthwhile to explore complex behavior made with the simplest rules and such models are called cellular automata.

A cellular automaton (CA) is a model of a system of "cell" objects with the following characteristics.

- The cells live on a ***grid***.
- Each cell has a ***state***. The number of state possibilities is typically finite. The simplest example has the two possibilities of 1 and 0 (otherwise referred to as "on" and "off" or "alive" and "dead").
- Each cell has a ***neighborhood***. This can be defined in any number of ways, but it is typically a list of adjacent cells.

off	off	on	off	on	on
on	off	off	off	on	on
on	off	on	on	on	off
off	off	on	off	on	on
on	on	off	off	on	off
on	on	on	off	off	on
on	off	off	on	on	on
off	off	on	off	on	off

Figure a: Neighborhood of cells in Conway's Game of Life

In 1970, Martin Gardner wrote an article in Scientific American that documented mathematician John Conway's new "Game of Life," describing it as "recreational" mathematics. It is a 2D CA with three simple rules: Live cells with 2 live neighbors survive, any cell with 3 neighbors becomes alive by reproduction and all the rest die from either loneliness or overpopulation. These simple rules produce a myriad of interesting patterns, such as those with move over boards, guns that fire bullets, spaceships, honey bee hives etc.

Perhaps the most significant scientific (and lengthy) work studying cellular automata arrived in 2002: Stephen Wolfram's 1,280-page "A New Kind of Science". Wolfram's book discusses how CA are not simply neat tricks, but are relevant to the study of biology, chemistry, physics, and all branches of science. Wofram's Elementary CA is able to produce complex structures like that of Serpinski's triangle, fractals and seemingly random patterns and may be used to generate random numbers. This project barely will scratch the surface of the theories Wolfram outlined (we will focus on the code implementation).

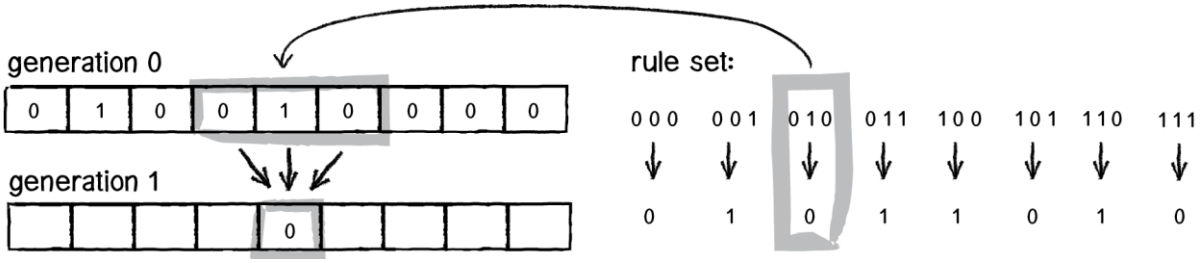


Figure b: Determining fate of the cells in Wolfram's Elementary CA (Rule 90)

CODE

```
01. //Libraries
02. #include <conio.h>
03. #include <iostream.h>
04. #include <ctype.h>
05. #include <iomanip.h>
06. #include <math.h>
07.
08. //Matrix constants
09. #define height 42
10. #define width 78
11. #define heightS 21
12.
13. //Characters for Border
14. #define BTMRC ((char) 217)
15. #define BTMLC ((char) 192)
16. #define TOPRC ((char) 191)
17. #define TOPLC ((char) 218)
18. #define VERTB ((char) 179)
19. #define HORIB ((char) 196)
20.
21. //Characters for cells
22. #define TOPC ((char) 223)
23. #define BTMC ((char) 220)
24. #define BOTH ((char) 219)
25. #define NONE ' '
26.
27. //Menu
28. #define MENU ((char) 240)
29. #define NOMENU 0
30. #define MENU1 1
31. #define MENUC 2
32. #define MENUW 3
33. #define ABOUT 4
34. #define HELPM 5
35.
36. //Pattern for Game of Life
37. #define GOSPER 1
38. #define GALAXY 2
39. #define K5 3
40. #define QUEEN 4
41. #define FIGURE 8
42. #define WEEK 7
43. #define CROSS 5
44.
45.
46. //Represents the pointer on the screen
47. struct Pointer {
48.   int x;
49.   int y;
50.   int justEdited;
51. } pointer;
52.
53.
54. //Initializations
55. int cells [2] [height] [width];        //Cell Board
56. char screenCells [heightS] [width];    //Screen Cells
57. int c = 0;                            //Current board
58. int conway = 1;                       //Is Conway's CA Active
59. int menu = NOMENU;                    //Current Menu
60. long int generations = 0;             //Generations count
61. int ruleD = 150, ruleB[8];            //Rules for Wolfram CA
62. int justInput = 0;                    //Smoothes input for Wolfram Rule
63.
64.
65.
66. //Saved Patterns
67. /*************************************************************/
```

```
//GOSPER
const int gosperH = 9, gosperW = 36;
int gosperP[gosperH][gosperW] =
        { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,
                  0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,
                  1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
                  1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,1,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };
//WEEKENDER
const int weekH = 16, weekW = 11;
int weekP[weekH][weekW] =
        { 0,0,0,0,0,0,0,0,1,0,0,
                0,0,0,0,0,0,1,1,0,1,1,
                0,0,0,1,0,1,0,0,1,0,0,
                0,0,0,1,0,0,0,0,0,0,0,
                0,1,0,1,0,0,0,0,0,0,0,
                1,0,0,1,0,0,0,0,0,0,0,
                1,0,0,0,1,1,0,0,0,0,0,
                0,0,0,0,1,1,0,0,0,0,0,
                0,0,0,0,1,1,0,0,0,0,0,
                1,0,0,0,1,1,0,0,0,0,0,
                1,0,0,1,0,0,0,0,0,0,0,
                0,1,0,1,0,0,0,0,0,0,0,
                0,0,0,1,0,0,0,0,0,0,0,
                0,0,0,1,0,1,0,0,1,0,0,
                0,0,0,0,0,0,1,1,0,1,1,
                0,0,0,0,0,0,0,0,1,0,0};

//QUEEN BEE SHUTTLE
const int queenH = 7, queenW = 22;
int queenP[queenH][queenW] =
       { 0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
               1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,
               1,1,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,1,1,
               0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0 };

//58P5H1V1
const int k5H = 23, k5W = 23;
int k5P[k5H][k5W] =
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,
             0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,
             0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,
             0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,1,0,0,1,
             0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
             0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,1,
             0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,
             0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,1,1,0,
             0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,
             0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,
             0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
             0,0,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
             0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
             0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
             0,0,1,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
             1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
             1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
            0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

//FIGURE 8
const int figureH = 6, figureW = 6;
int figureP[figureH][figureW] =
        { 1,1,0,0,0,0,
                  1,1,0,1,0,0,
                  0,0,0,0,1,0,
                  0,1,0,0,0,0,
                  0,0,1,0,1,1,
                  0,0,0,0,1,1};

//KOK'S GALAXY
const int galaxyH = 9, galaxyW = 9;
int galaxyP[galaxyH][galaxyW] =
        { 0,0,1,0,0,1,0,1,0,
                  1,1,0,1,0,1,1,1,0,
                  0,1,0,0,0,0,0,0,1,
                  1,1,0,0,0,0,0,1,0,
                  0,0,0,0,0,0,0,0,0,
                  0,1,0,0,0,0,0,1,1,
                  1,0,0,0,0,0,0,1,0,
                  0,1,1,1,0,1,0,1,1,
                  0,1,0,1,0,0,1,0,0};


//CROSS
const int crossH = 8, crossW = 8;
int crossP[crossH][crossW] =
           { 0,0,1,1,1,1,0,0,
                   0,0,1,0,0,1,0,0,
                   1,1,1,0,0,1,1,1,
                   1,0,0,0,0,0,0,1,
                   1,0,0,0,0,0,0,1,
                   1,1,1,0,0,1,1,1,
                   0,0,1,0,0,1,0,0,
                   0,0,1,1,1,1,0,0};

/*****************************************************************/

//Initializes the screen cells
void initializeScreen() {
  for (int h = 0; h < heightS; ++h)
    for (int w = 0; w < width; ++w)
      screenCells[h][w] = NONE;
}


//Initializes all cells for 2 boards to 0
void initializeCells() {
  generations = 0;
  for (int j = 0; j < height; ++j)
    for (int k = 0; k < width; ++k)
      cells[0][j][k] = cells[1][j][k] = 0;
}


//Inserts an array of char into a row of the screenCells
void insert(char c[], int row) {
  for (int i = 0; (c[i] != '\0') && (i < 77); ++i)
    screenCells[row][i] = c[i];
}


void screenMenu() {
  initializeScreen();

  //Printing main menu
```

```
  if (menu == MENU1) {
    char text1[77] = "  (1) Change Cellular Automaton ";
    char text2[77] = "  (2) Load a 'Game of Life' pattern";
    char text3[77] = "  (3) Load an 'Elementary CA' Rule No.";
    char text4[77] = "  (4) About Cellular Automata";
    char text5[77] = "  (5) Help on Controls";
    char text6[77] = "  << Press M to exit menu";
    char text7[77] ="Program by Harsh Vardhan Singh, Ishvik Kumar Singh and Utsav Munendra";
    insert(text1,2); insert(text2,4); insert(text3,5); insert(text4,7); insert(text5,8);
    insert(text6,18); insert(text7,20);
  }

  //Printing About Menu
  else if (menu == ABOUT) {
    char text2[77]="We exist in a complex world, a universe governed by the complex rules of   ";
    char text3[77]="all sciences. In computational world, it happens that complex systems are  ";
    char text4[77]="relatively easy to achieve, as evident by the existance of millions of     ";
    char text5[77]="video games and virtual environments. But in cellular automata, we explore ";
    char text6[77]="the 'simplest' complex systems which are governed by the simplest rules.   ";
    char text7[77]="By definition, a complex system is a system of many simple agents that work";
    char text8[77]="together to exhibit complex, intelligent behavior.                         ";
    char text9[77]="Stephen Wolfram's Elementary CA is one of most recent and major 1D CA. By  ";
    char text10[77]="stacking successive generations over one another, various patterns emerge  ";
    char text11[77]="which can be classified as uniform, oscillating, random and complex. The   ";
    char text12[77]="next generation is computed through the rule which is a number from 0-255  ";
    char text13[77]="John Conway's Game of Life is a 2D CA with the fate of cells in the grid   ";
    char text14[77]="depending upon the number of neighbours around it. Cells with 3 neighbours,";
    char text15[77]="whether dead or alive, come to life by reproduction, and live cells with 2 ";
    char text16[77]="neighbours survive. All rest die from underpopulation or crowding.         ";
    char text18[77]="<< Press M to go to Main Menu                                             ";
    insert(text2,1); insert(text3,2); insert(text4,3); insert(text5,4); insert(text6,5);
    insert(text7,6); insert(text8,7); insert(text13,9); insert(text14,10); insert(text15,11);
    insert(text16,12); insert(text9,14); insert(text10,15); insert(text11,16);
    insert(text12,17); insert(text18,20);
  }

  //Printing Help menu
  else if (menu == HELPM) {
    char text1[30]  = "  (W)    Move pointer up";
    char text2[30]  = "  (A)    Move pointer left";
    char text3[30]  = "  (S)    Move pointer right";
    char text4[30]  = "  (D)    Move pointer down";
    char text5[30]  = "  (C)    Clear cell grid";
    char text6[30]  = "  (K)    Make cell alive/dead";
    char text7[30]  = "  (X)    Exit program";
    char text8[30]  = "  (M)    Control menu";
    char text9[30]  = "  Space  Next generation";
    char text10[30] = "  1-9    Choose menu options";
    char text11[40] = "  << Press M to go back to Main Menu";
    insert(text1,2); insert(text2,3); insert(text3,4); insert(text4,5);
    insert(text5,7); insert(text6,8); insert(text7,9); insert(text8,10);
     insert(text9,12); insert(text10,13); insert(text11,20);
  }

  //Printing Wolfram's Menu
  else if (menu == MENUW) {
    char text1[60]  = "  Enter Rule Number Below:";
    char text2[60]  = "  Numbers outside 0-255 inclusive range will";
    char text3[60]  = "  matched to a rule inside the range.";
    char text4[60]  = "  Interesting Rules";
    char text5[60]  = "  Serpinski's Triangle: 18, 22, 126, 129, 181";
    char text6[60]  = "  Fractal Triangles: 60, 110";
    char text7[77]  = "  Complex Triangles: 30, 57, 62, 73, 75, 101, 105, 109, 150 ";
    char text8[60]  = "  Complex Patterns: 45, 89, 107 ";
    char text9[60]  = "  Other: 54, 99, 250" ;
    char text11[40] = "  << Press M to go back to Main Menu ";
    insert(text1,2); insert(text2,4); insert(text3,5); insert(text4,8);
    insert(text8,9); insert(text7,10); insert(text5,11); insert(text6,12);
```

```
    insert(text9,13); insert(text11,20);
  }

  //Printing Conway's Menu
  else if (menu == MENUC) {
    char text1[60]  = "  Choose among the following:";
    char text2[60]  = "  (1) Gosper Glider Gun";
    char text3[60]  = "  (2) Weekender";
    char text4[60]  = "  (3) Queen Bee Shuttle";
    char text5[60]  = "  (4) Figure 8";
    char text6[60]  = "  (5) Kok's Galaxy";
    char text7[77]  = "  (6) Cross";
    char text8[60]  = "  (7) 58P5H1V1";
    char text11[40] = "  << Press M to go back to Main Menu ";
    insert(text1,2); insert(text2,4); insert(text3,5); insert(text4,6);
    insert(text5,7); insert(text6,8); insert(text7,9); insert(text8,10);
    insert(text11,20);
  }
}

//Loads a Conway CA pattern on the board
void load(int pattern) {
  menu = NOMENU;
  conway = 1;
  int h,w;

  switch(pattern) {
  int h,w,i,j;
    case GOSPER:                        //Gosper Glider Gun
      h = gosperH;
      w = gosperW;
      for (i = 0; i < h; ++i) {
    int next;
    int down = (pointer.y+i)%height;
    for (int j = 0; j < w; ++j) {
      next = (pointer.x+j)%width;
      cells[c][down][next] = gosperP[i][j];
    }
    next = pointer.x;
    down = pointer.y;
      }
    break;

    case WEEK:                          //Weekender
      h = weekH;
      w = weekW;
      for (i = 0; i < h; ++i) {
    int next;
    int down = (pointer.y+i)%height;
    for (int j = 0; j < w; ++j) {
      next = (pointer.x+j)%width;
      cells[c][down][next] = weekP[i][j];
    }
    next = pointer.x;
    down = pointer.y;
      }
    break;
    case QUEEN:                         //Queen Bee Shutler
      h = queenH;
      w = queenW;
      for (i = 0; i < h; ++i) {
    int next;
    int down = (pointer.y+i)%height;
    for (int j = 0; j < w; ++j) {
      next = (pointer.x+j)%width;
      cells[c][down][next] = queenP[i][j];
    }
    next = pointer.x;
```

```
    down = pointer.y;
      }
    break;
    case GALAXY:                          //Kok's Galaxy
      h = galaxyH;
      w = galaxyW;
      for (i = 0; i < h; ++i) {
    int next;
    int down = (pointer.y+i)%height;
    for (int j = 0; j < w; ++j) {
      next = (pointer.x+j)%width;
      cells[c][down][next] = galaxyP[i][j];
    }
    next = pointer.x;
    down = pointer.y;
      }
    break;

    case CROSS:                           //Cross
      h = crossH;
      w = crossW;
      for (i = 0; i < h; ++i) {
    int next;
    int down = (pointer.y+i)%height;
    for (int j = 0; j < w; ++j) {
      next = (pointer.x+j)%width;
      cells[c][down][next] = crossP[i][j];
    }
    next = pointer.x;
    down = pointer.y;
      }
    break;

    case FIGURE:                          //Figure 8
      h = figureH;
      w = figureW;
      for (i = 0; i < h; ++i) {
    int next;
    int down = (pointer.y+i)%height;
    for (int j = 0; j < w; ++j) {
      next = (pointer.x+j)%width;
      cells[c][down][next] = figureP[i][j];
    }
    next = pointer.x;
    down = pointer.y;
      }
    break;
    case K5:                              //58P5H1V1
      h = k5H;
      w = k5W;
      for (i = 0; i < h; ++i) {
    int next;
    int down = (pointer.y+i)%height;
    for (int j = 0; j < w; ++j) {
      next = (pointer.x+j)%width;
      cells[c][down][next] = k5P[i][j];
    }
    next = pointer.x;
    down = pointer.y;
      }
    break;
  }
}


void printCells() {

  clrscr();
```

```
408.
409.    if (!menu) {
410.      //Determining pointer characteristics
411.      int up = (pointer.y % 2 == 0)? 1 : 0;
412.
413.      //Translating board cells to screen cells
414.      for (int r = 0; r < height; r+=2) {
415.        for (int w = 0; w < width; ++w) {
416.      int r2 = r/2;
417.      if (cells[c][r][w] && cells[c][r+1][w])
418.        screenCells[r2][w] = BOTH;
419.      else if (cells[c][r][w] && !cells[c][r+1][w])
420.        screenCells[r2][w] = TOPC;
421.      else if (!cells[c][r][w] && cells[c][r+1][w])
422.        screenCells[r2][w] = BTMC;
423.      else screenCells[r2][w] = NONE;
424.
425.      //Placing pointer on the screen and hiding when board is just edited
426.      if (!pointer.justEdited && (pointer.y==r || pointer.y==r+1) && pointer.x==w) {
427.
428.        //Non-overlapping cell and pointer on same screen cell
429.        if ((up && screenCells[r2][w] == BTMC) ||
430.           (!up && screenCells[r2][w] == TOPC))
431.          screenCells[r2][w] = BOTH;
432.
433.        //Overlapping cell and pointer in same screen cell
434.        else if ((up && screenCells[r2][w] == TOPC) ||
435.           (!up && screenCells[r2][w] == BTMC))
436.          screenCells[r2][w] = NONE;
437.
438.        //Two cells and the pointer in the same screen cell
439.        else if (screenCells[r2][w] == BOTH)
440.          if (up) screenCells[r2][w] = BTMC;
441.          else screenCells[r2][w] = TOPC;
442.
443.        //Only pointer in the screen cell
444.        else if (screenCells[r2][w] == NONE)
445.          if (up) screenCells[r2][w] = TOPC;
446.          else screenCells[r2][w] = BTMC;
447.      }
448.        }
449.      }
450.    }
451.    //If a menu is active, that menu is printed
452.    else screenMenu();
453.
454.    //Printing Header
455.    textbackground(7);
456.    textcolor(RED);
457.    cprintf("  ");
458.    cprintf("%c ", MENU);
459.    if (menu) textbackground(3);
460.    cprintf(" M");
461.    textcolor(BLACK);
462.    cprintf("enu ");                          //Menu
463.    textbackground(7);
464.    cprintf("                  ");
465.    textcolor(0);
466.    textbackground(14);                       //Title
467.    if (!menu) {
468.      if (conway)
469.        cprintf("  Conway's Game of Life  ");
470.      else
471.        cprintf(" Wolfram's Elementary CA ");
472.    }
473.    else if (menu == ABOUT) cprintf("    Cellular Automata    ");
474.    else if (menu == MENU1) cprintf("        Main Menu        ");
475.    else if (menu == HELPM) cprintf("     Program Controls    ");
```

```
  else if (menu == MENUW) cprintf("    Enter Wolfram Rule   ");
  else if (menu == MENUC) cprintf("Load Conway Configuration");
  textbackground(7);
  cprintf("                    E");        //Exit
  textcolor(RED);
  cprintf("x");
  textcolor(BLACK);
  cprintf("it  ");
  textbackground(BLACK);
  textcolor(7);

  //Printing screen and the box
  for (int h = -1; h <= heightS; ++h)
    for (int w = -1; w <= width; ++w) {
      if (h==-1) {
    if (w==-1)  cout << TOPLC;
    else if (w==width) cout << TOPRC;
    else cout << HORIB;
      }
      else if (h==heightS) {
    if (w==-1)  cout << BTMLC;
    else if (w==width) cout << BTMRC;
    else cout << HORIB;
      }
      else if (w==-1 || w==width) cout << VERTB;
      else cout << screenCells[h][w];
  }

  //Printing footer
  cout << "  Generation: " << setw(6) << generations;
  if (conway && !menu)
    cout << "                    ";
  else if (!menu)
    cout << "           Rule: " << setw(3) << ruleD;

  if (!menu) {
    //Printing Conway CA controls
    if (conway)
      cout << "                  W " << (char) 30 <<"  A "
       << ((char) 17) << "  S " << ((char) 31) << "  D "
       << ((char) 16) << " ";

    //Printing  Wolfram CA controls
    else
      cout << "                              A " << (char) 17 <<"  D "
       << ((char) 16);
  }
  else if (menu == MENUW) {
    cout << "\tRule: ";      //Printing the rule for input if
  }                          //Wolfram Menu is active
}


//Returns number of Moore neighbours of a cell
int getMooreNeighbours(int c1, int h, int w) {

  int left  = ((w <= 0)? width-1 : w-1);        //Boundary wrapping
  int right = ((w >= width-1)? 0 : w+1);
  int top   = ((h <= 0)? height-1: h-1);
  int bottm = ((h >= height-1)? 0 : h+1);

  return (cells[c1][top][left] + cells[c1][top][w] + cells[c1][top][right] +
      cells[c1][h][left] + cells[c1][h][right] +
      cells[c1][bottm][left] + cells[c1][bottm][w] + cells[c1][bottm][right]);
}


//Converts decimal Wolfram rule to its binary equivalent
```

```
544.  void ruleD_B() {
545.    int D = ruleD;
546.    for (int i = 7; i >= 0; --i) {
547.      if (pow(2,i) <= D) {
548.        D -= pow(2,i);
549.        ruleB[i] = 1;
550.      } else ruleB[i] = 0;
551.    }
552.  }
553.
554.
555.  void nextConwayGen() {
556.    int p = c;        //Index of previous board
557.    c = !c;           //Current board changed
558.    ++generations;
559.
560.    //Computing the fate for every cell
561.    for (int h = 0; h < height; ++h)
562.      for (int w = 0; w < width; ++w) {
563.        int num = getMooreNeighbours(p,h,w);       //Getting neighbours
564.
565.        //John Conway's Rules
566.        cells[c][h][w]=0;
567.        if ((cells[p][h][w] && num==2) || (num==3))
568.      cells[c][h][w] = 1;
569.      }
570.  }
571.
572.
573.  void nextWolframGen() {
574.    ++generations;
575.
576.    //Shifting previous generations up
577.    for (int r = 0; r < height-1; ++r)
578.      for (int w = 0; w < width; ++w)
579.        cells[c][r][w] = cells[c][r+1][w];
580.
581.    //Computing current generation
582.    for (int i = 0; i < width; ++i) {
583.      int row = height-1;
584.      int left = (i==0)? width-1: i-1;         //Boundary Wrapped
585.      int right= (i==width-1)? 0: i+1;
586.      int n = (4*cells[c][row-1][left]) +      //Cell state converted
587.          (2*cells[c][row-1][i]) +         //to an index in the binary
588.             cells[c][row-1][right];       //rule array.
589.
590.      //Determining cell state from rule
591.      cells[c][row][i] = ruleB[n];
592.    }
593.  }
594.
595.
596.  void main() {
597.    clrscr();
598.    //Initializations
599.    char input = '-';
600.    pointer.x = width/2;            //Pointer initial locations
601.    pointer.y = height/2;
602.    c = pointer.justEdited = 0;
603.    ruleD_B();
604.    cout << "\n\n\n\n\n\n\n\n\n\n\n\t\t\t    Press any key to start";
605.
606.    //Program exit controller
607.    while (input != 'x') {
608.
609.      //Input from user
610.      if (!justInput) input = tolower(getch());
611.      else {
```

```
612.        input = '-';         //For smooth exit from Wolfram Menu
613.        justInput = 0;
614.      }
615.
616.      if (!menu) {
617.        //Displaying board
618.        switch (input) {
619.      case 'w':                                          //Up
620.        if (conway) {
621.          if (--pointer.y < 0) pointer.y = height-1;
622.          pointer.justEdited = 0;
623.        }
624.        break;
625.      case 'a':                                          //Left
626.        if (--pointer.x < 0) pointer.x = width-1;
627.        pointer.justEdited = 0;
628.        break;
629.      case 's':
630.        if (conway) {                                    //Down
631.          if (++pointer.y >= height) pointer.y = 0;
632.          pointer.justEdited = 0;
633.        }
634.        break;
635.      case 'd':                                          //Right
636.        if (++pointer.x >= width) pointer.x = 0;
637.        pointer.justEdited = 0;
638.        break;
639.      case 'k':                                          //Cell
640.        cells[c][pointer.y][pointer.x] = !cells[c][pointer.y][pointer.x];
641.        pointer.justEdited = 1;
642.        break;
643.      case ' ':                                          //Next Generation
644.        (conway)? nextConwayGen(): nextWolframGen();
645.        break;
646.      case 'c':                                          //Clear Board
647.        initializeCells();
648.        break;
649.      case 'm':                                          //Display Menu 1
650.        menu = MENU1;
651.        break;
652.        }
653.      }
654.      //Displaying Menu 1
655.      else if (menu == MENU1) {
656.      switch (input) {
657.        case 'm':                               //Exit Menu
658.          menu = NOMENU;
659.          break;
660.        case '1':                               //Change CA
661.          conway = !conway;
662.          menu = NOMENU;
663.          initializeCells();
664.          pointer.y = height-1;
665.          break;
666.        case '2':                               //Enter Conway pattern
667.          menu = MENUC;
668.          break;
669.        case '3':                               //Enter Wolfram Rule
670.          menu = MENUW;
671.          break;
672.        case '5':                               //Help Menu
673.          menu = HELPM;
674.          break;
675.        case '4':                               //About CA
676.          menu = ABOUT;
677.          break;
678.      }
679.      }
```

```
    //Displaying Conway Menu
    else if (menu == MENUC) {
      switch(input) {
    case 'm':                                   //Exit menu
      menu = MENU1;
      break;
    case '1':                                   //Load Gosper Glider Gun
      load(GOSPER);
      break;
    case '2':                                   //Load Weekender
      load(WEEK);
      break;
    case '3':                                   //Load Queen Bee Shutler
      load(QUEEN);
      break;
    case '4':                                   //Load Figure 8
      load(FIGURE);
      break;
    case '5':                                   //Load Kok's Galaxy
      load(GALAXY);
      break;
    case '6':                                   //Load Cross
      load(CROSS);
      break;
    case '7':                                   //Load 58P5H1V1
      load(K5);
      break;
      }
    }
    //Controlling other menu's exit
    else {
      if (input == 'm')
    menu = MENU1;
    }
    //Printing the screen
    printCells();

    //Input for Wolfram Rule
    if (menu == MENUW) {
      cin >> ruleD;
      menu = NOMENU;                    //Menu exit
      justInput = 1;
      conway = 0;                       //Wolfram Actived
      initializeCells();                //Board Initialized
      pointer.y = height-1;             //Pointer relocated
      if (ruleD < 0)   ruleD = -ruleD;  //Error Check
      if (ruleD > 256) ruleD %= 256;
      ruleD_B();                        //Decimal Rule to Binary
    }
  }
}
```

OUTPUT

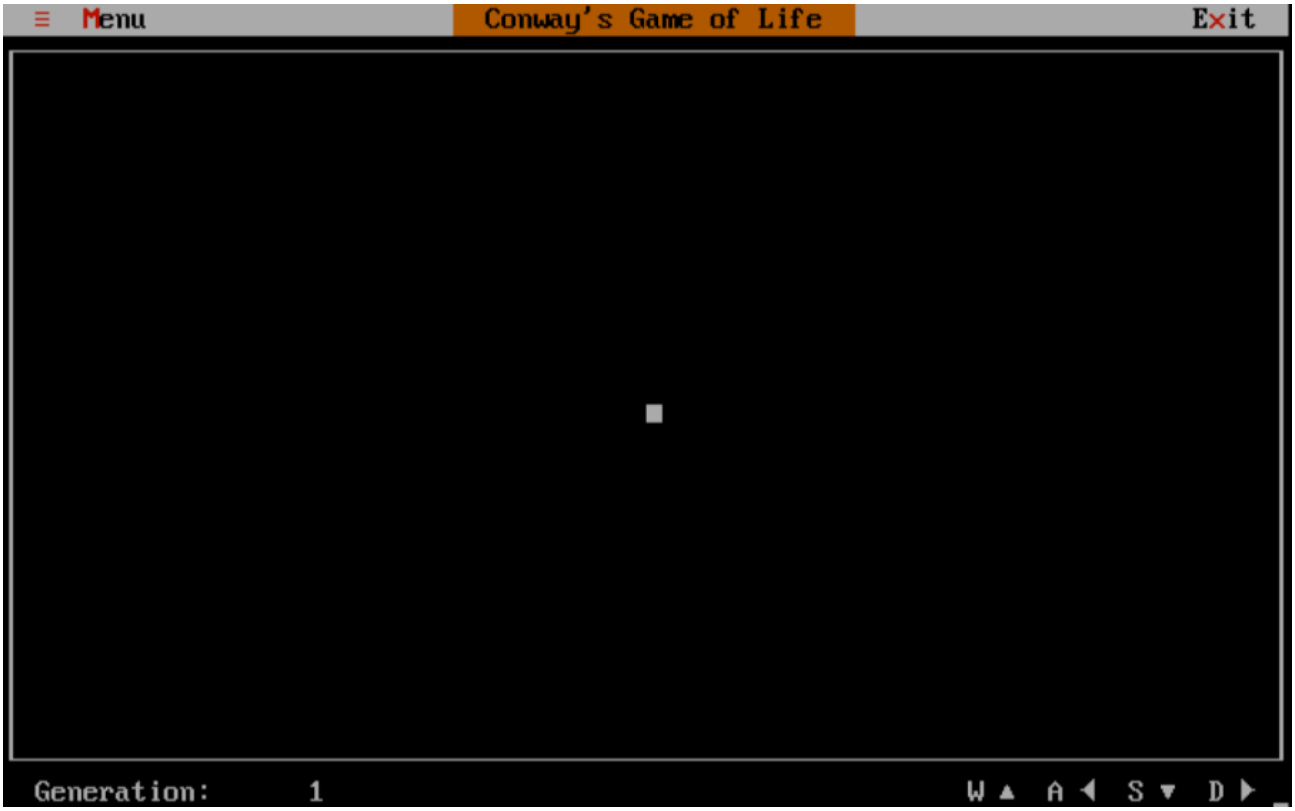


Figure 1: User Interface

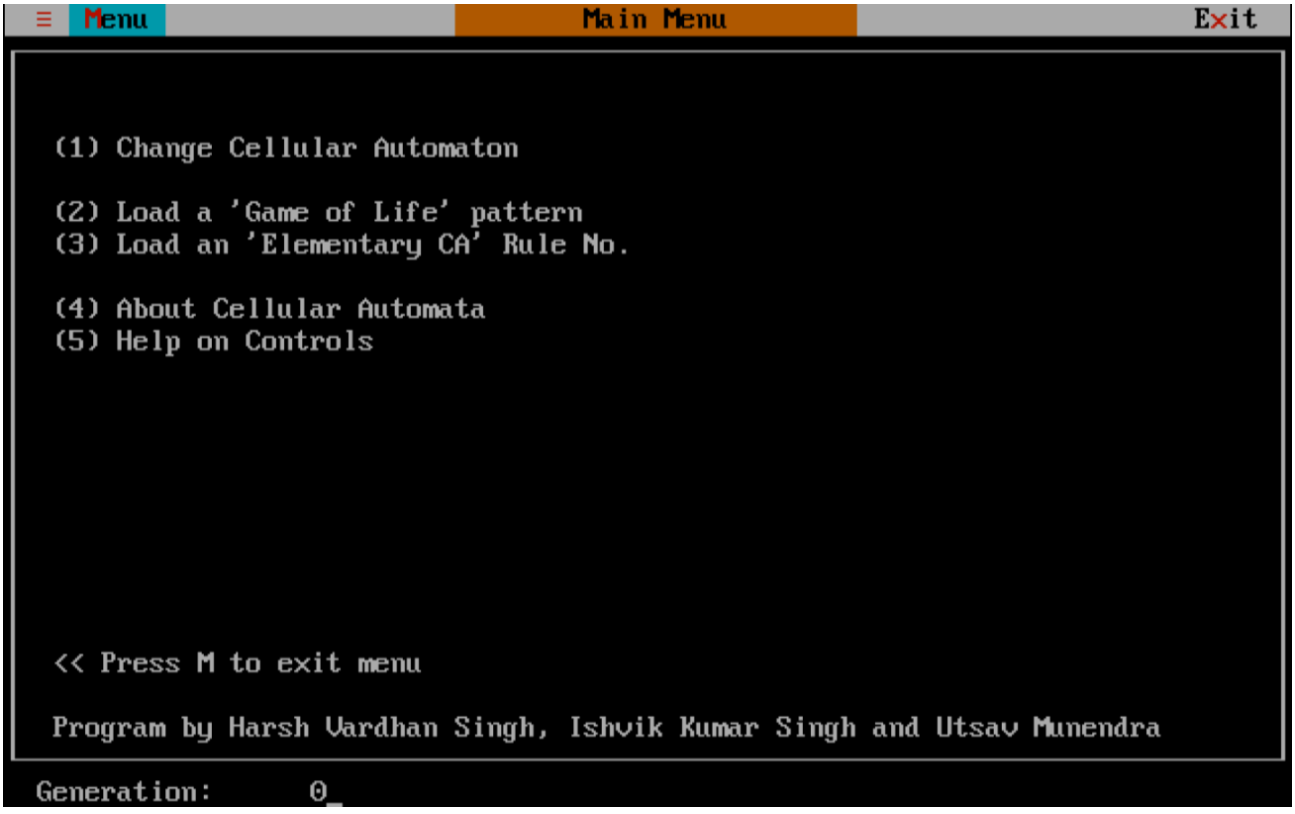


Figure 2: Main Menu

Figure 3:
About Cellular Automata

Information about cellular automata

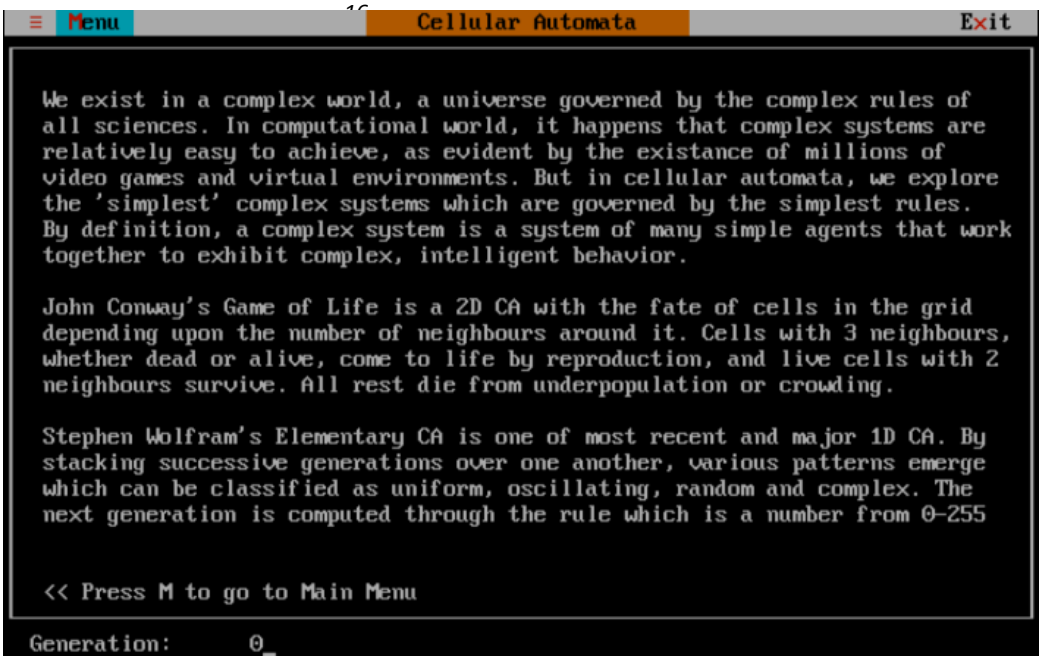


Figure 4:
Conway's Menu

Loads any of given Conway configurations onto the board, wherever the pointer is.

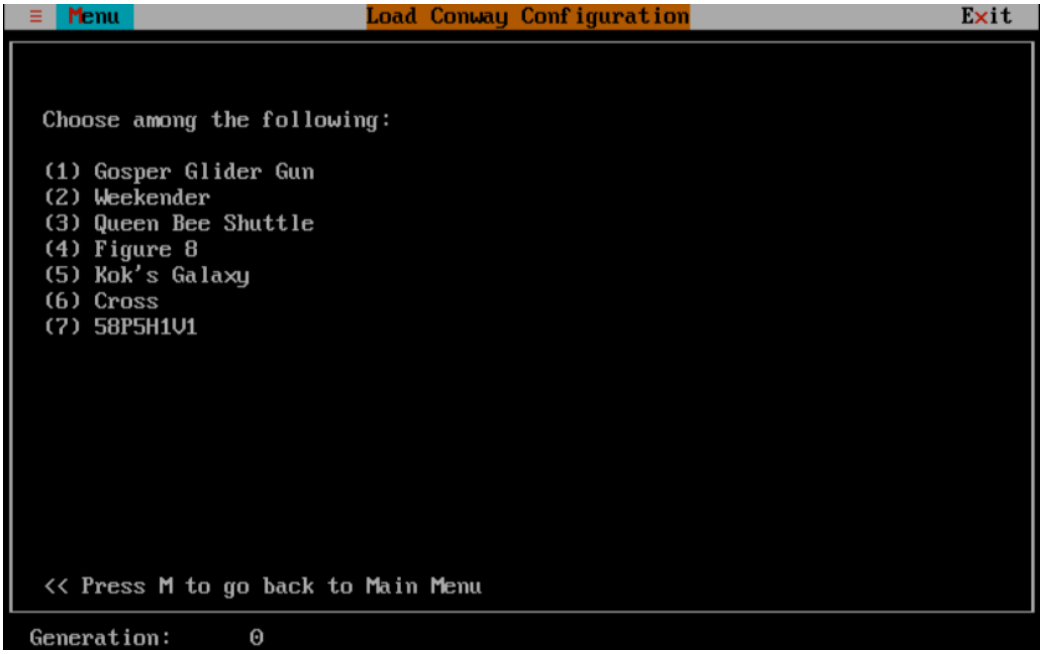
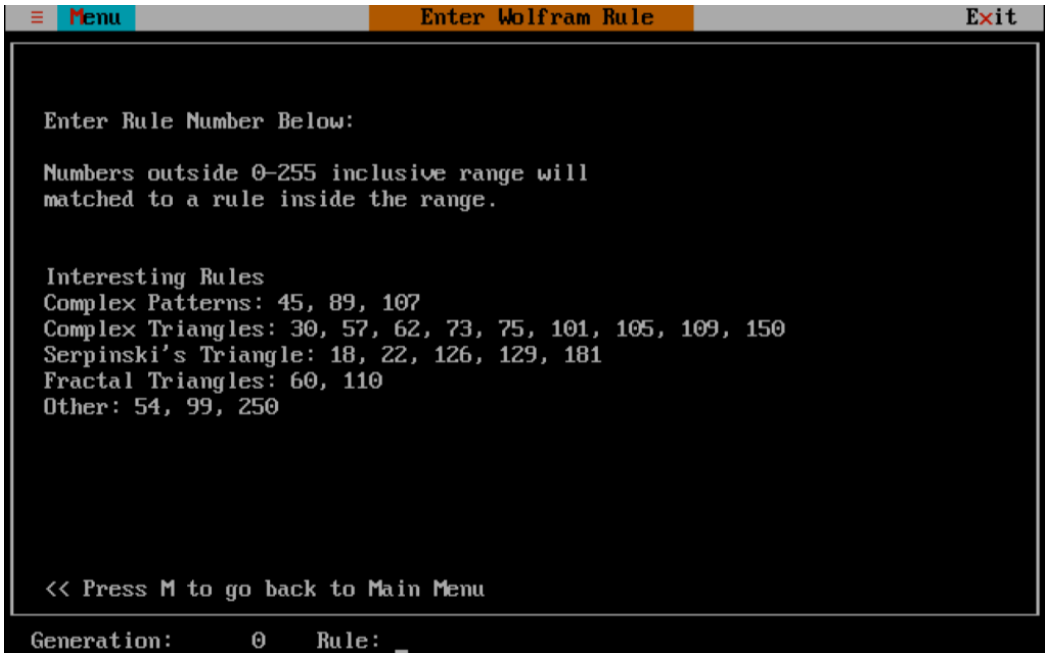


Figure 5:
Wolfram's Menu

Change the rule for Wolfram Elementary CA. Some of the rules which give rise to complex patterns are also presented.



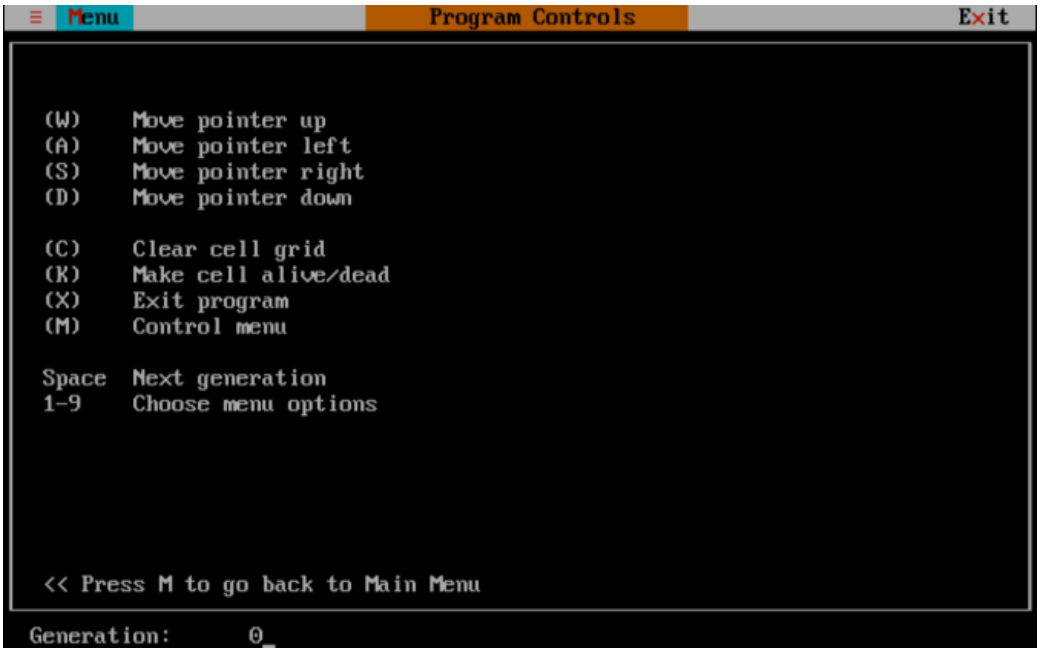


Figure 6:
Program Controls

Instruction Guide for running the program

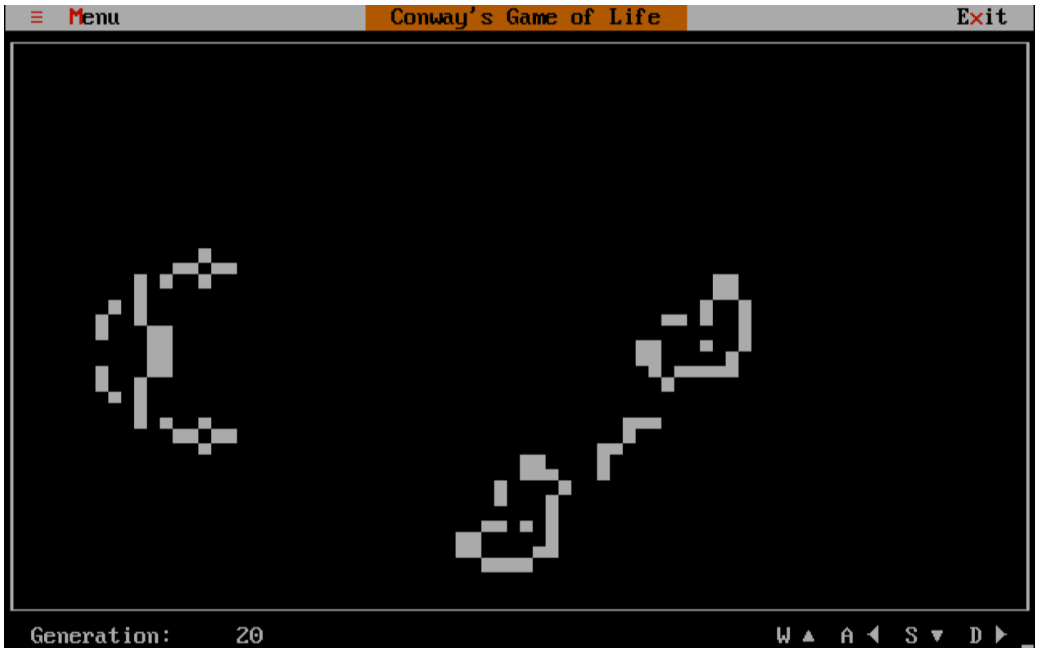


Figure 7:
Game of Life

'Weekender' and '58P5H1V1' patterns loaded on the board to collide.

Figure 8:
Game of Life

The collision took nearly 1140 generations to stabilize. Two gliders emerged in between and this finally resulted in 5 still lives and 2 oscillators.

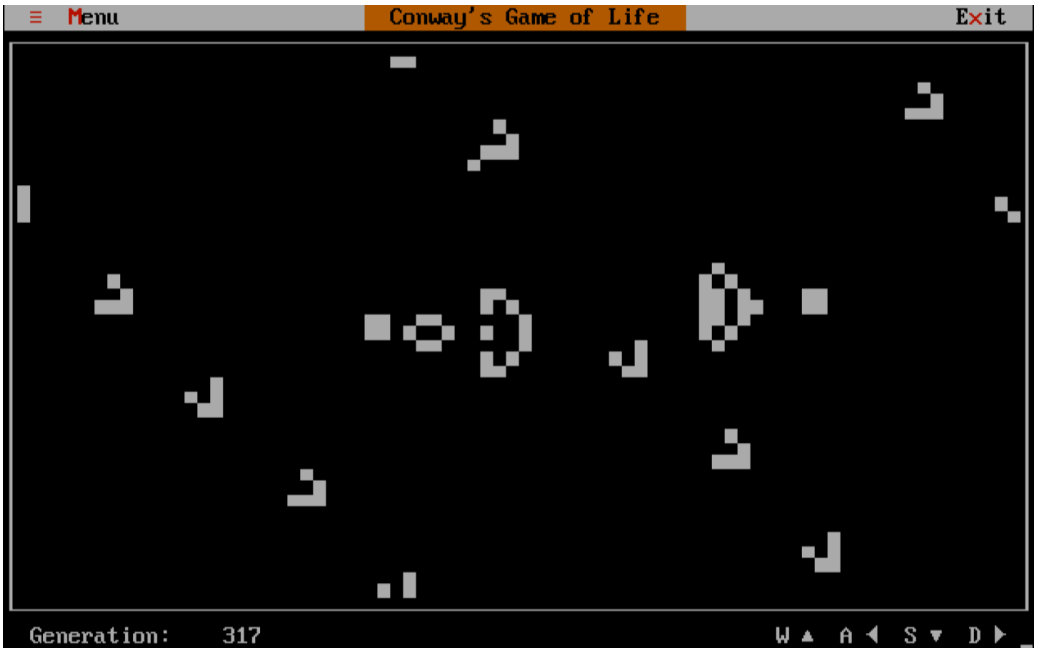


Figure 9:
Game of Life
Gosper Glider Gun

The image shows 'Gosper Glider Gun' pattern after being executed 317 times. This pattern endlessly generated gliders which finally collide back to the gun in a board with wrapped ends.

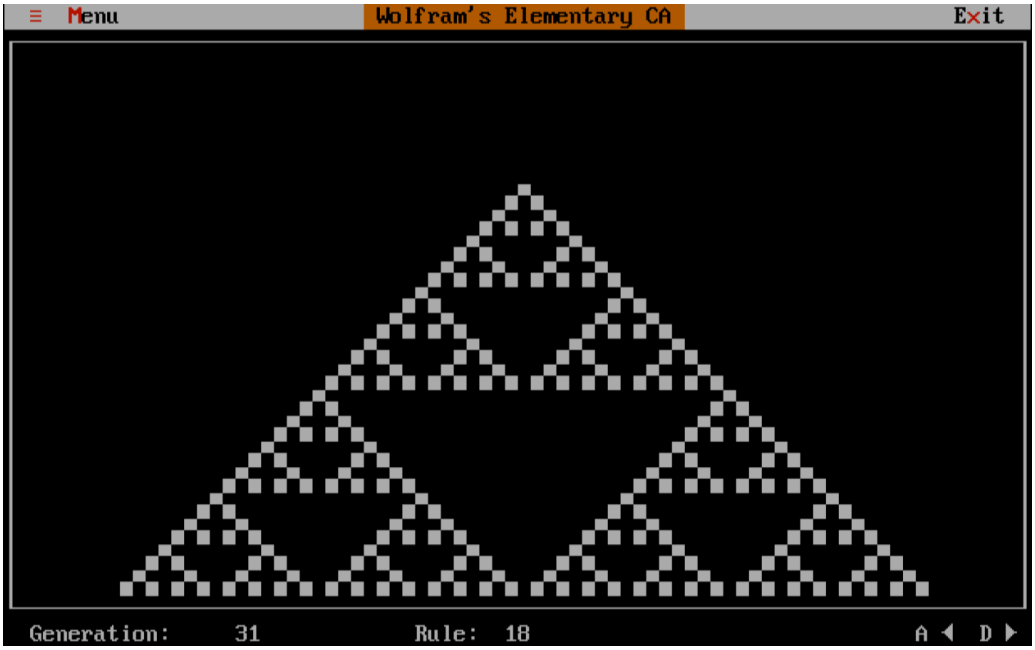


Figure 10:
Wolfram's Elementary CA
Rule 18

Rule 18 generates a Serpinski's Triangle which is a triangle made with three smaller Serpinki's triangles.

It is remarkable how the simple rule can demonstrate powerful concepts of recursion.

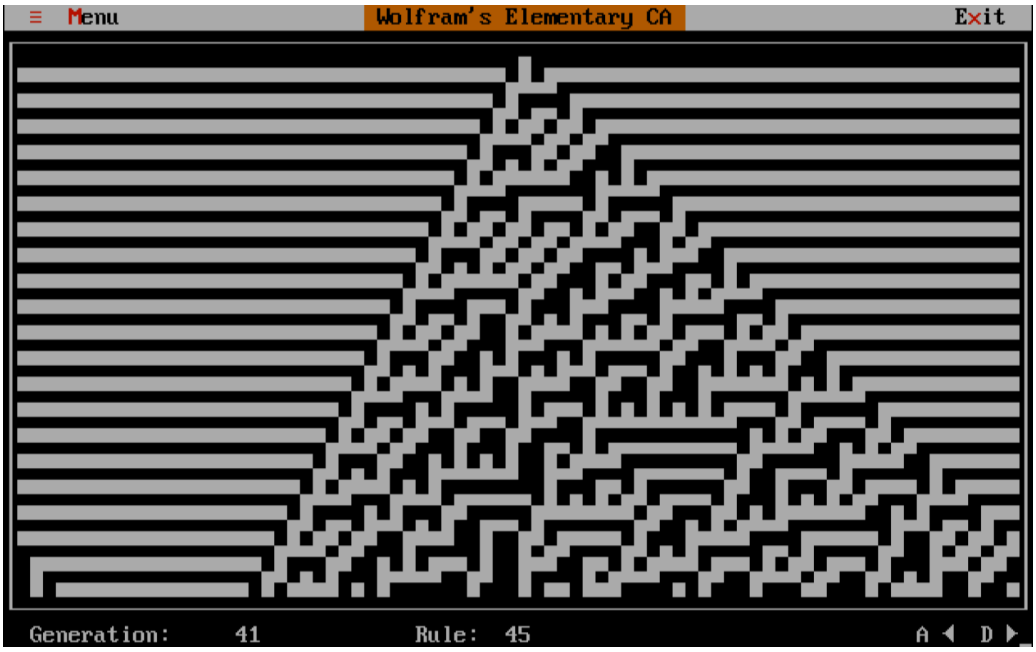


Figure 11:
Wolfram's Menu
Rule 45

This rule generates an apparently random structures even though it involves predictable calculations.

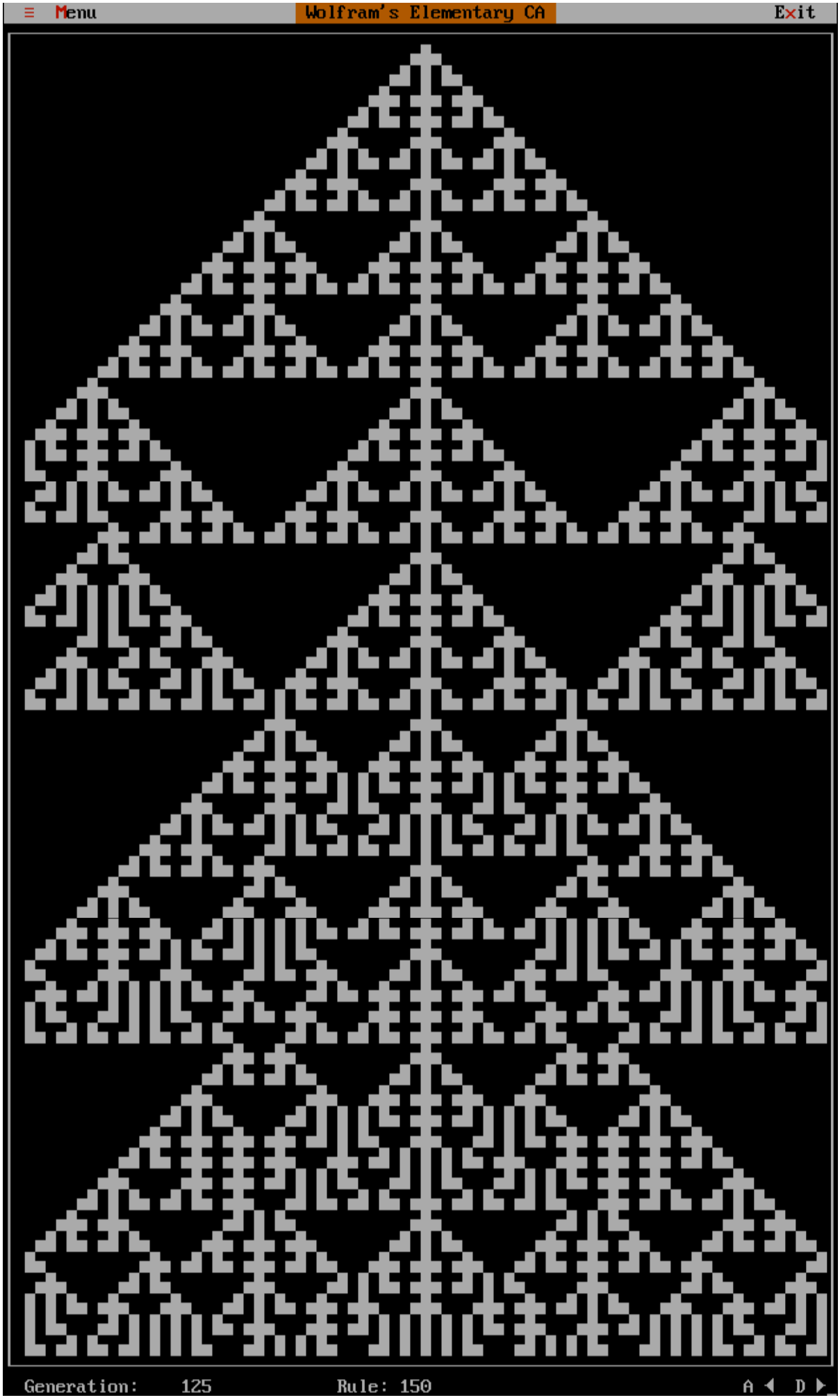


Figure 12: **Rule 150:** *A Complex Patter, oscillating yet random.*

BIBLIOGRAPHY

Shiffman, Daniel (2012). *"The Nature of Code"*
From www.natureofcode.com/book/chapter-7-cellular-automata/

Weisstein, Eric W. ***"Elementary Cellular Automaton"***
From www.mathworld.wolfram.com/ElementaryCellularAutomaton

Arora, Sumita (2016). *"Computer Science with C++"*