VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI

Mini Project Report on

"VISUALIZATION OF CELLULAR AUTOMATA"

Submitted in the partial fulfillment for the requirements of Computer Graphics & Visualization Laboratory of 6th semester CSE requirement in the form of the Mini Project work

Submitted By

M AISHWARYA **USN: 1BY17CS085**

M MERLYN MERCYLONA **USN: 1BY17CS088**

Under the guidance of

Mr. SHANKAR R
Assistant Professor, CSE, BMSIT&M

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT
YELAHANKA, BENGALURU - 560064.

2019-2020

BMS INSTITUTE OF TECHNOLOGY &MANAGEMENT

YELAHANKA, BENGALURU – 560064

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Project work entitled **"VISUALIZATION OF CELLULAR AUTOMATA"** is a bonafide work carried out by **M AISHWARYA (1BY17CS085) and M MERLYN MERCYLONA (1BY17CS088)** in partial fulfillment for *Mini Project* during the year 2019-2020. It is hereby certified that this project covers the concepts of *Computer Graphics & Visualization*. It is also certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in this report.

Signature of the Guide with date
Mr. SHANKAR R
Assistant Professor
CSE, BMSIT&M

Signature of HOD with date
Dr. Anil G N
Prof & Head
CSE, BMSIT&M

INSTITUTE VISION

To emerge as one of the finest technical institutions of higher learning, to develop engineering professionals who are technically competent, ethical and environment friendly for betterment of the society.

INSTITUTE MISSION

Accomplish stimulating learning environment through high quality academic instruction, innovation and industry-institute interface.

DEPARTMENT VISION

To develop technical professionals acquainted with recent trends and technologies of computer science to serve as valuable resource for the nation/society.

DEPARTMENT MISSION

Facilitating and exposing the students to various learning opportunities through dedicated academic teaching, guidance and monitoring.

PROGRAM EDUCATIONAL OBJECTIVES

1. Lead a successful career by designing, analyzing and solving various problems in the field of Computer Science & Engineering.
2. Pursue higher studies for enduring edification.
3. Exhibit professional and team building attitude along with effective communication.
4. Identify and provide solutions for sustainable environmental development.

ACKNOWLEDGEMENT

We are happy to present this project after completing it successfully. This project would not have been possible without the guidance, assistance and suggestions of many individuals. We would like to express our deep sense of gratitude and indebtedness to each and every one who has helped us make this project a success.

We heartily thank our Principal, Dr. MOHAN BABU G N, BMS Institute of Technology &Management, for his constant encouragement and inspiration in taking up this project.

We heartily thank our Professor and Head of the Department, Dr. ANIL G N, Department of Computer Science and Engineering, BMS Institute of Technology &Management, for his constant encouragement and inspiration in taking up this project.

We gracefully thank our Project Guide, Mr. Shankar R, Assistant Professor, Department of Computer Science and Engineering for his intangible support and for being constant backbone for our project.

Special thanks to all the staff members of Computer Science Department for their help and kind co-operation.

Lastly, we thank our parents and friends for the support and encouragement given throughout in completing this precious work successfully.

M AISHWARYA (1BY17CS085)

M MERLYN MERCYLONA (1BY17CS088)

ABSTRACT

Cellular automata are a way of modelling pixels on screen. It can form complex patterns and animations from some simple rules, through various iterations. It is used to model systems in biology, physics, generate random numbers for cryptography, etc. One subpart of this is an algorithm called Conway's game of life. An initial configuration is created using pixels, and its evolution is observed. We will focus on implementing this algorithm in this project with OpenGL.

Conway's game of life has 3 simple rules that govern the status of an organism, in this case a pixel on screen. It has rules that create and destroy pixels, in a manner that leads to interesting patterns being formed. We implement this concept as an animation using OpenGL.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BRIEF INTRODUCTION:

OpenGL is a library for doing computer graphics. By using it, we can create interactive applications which render high-quality color images composed of 3D geometric objects and images. OpenGL is window and operating system independent.

However, in order for OpenGL to be able to render, it needs a window to draw into. Generally, this is controlled by the windowing system on whatever the platform we are working on. As OpenGL is platform independent, we need some way to integrate OpenGL into each windowing system. Every windowing system where OpenGL is supported has additional API calls for managing OpenGL windows, color maps and other features. These additional APIs are platform dependent. For the sake of simplicity, we are using an additional freeware library, GLUT for simplifying interaction with windows. GLUT, the OpenGL Utility Toolkit, is a library of utilities for OpenGL programs, which primarily performs system-level I/O with the host operating system

The goal of our project is to design a visualization of the Cellular Automata as described by John Horton Conway, using OpenGL in C. This Conway's Game of Life is meant to show what happens to organisms when they're placed in close proximity to each other. Upon giving the Game's initial conditions, each successive generation (iteration) shows the evolution of the organisms.

The board of game is meant to represent the ecosystem in which the organisms live in. In the Universe of the Conway's Game of Life representation, this is an infinite, two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead. Every cell interacts with its eight neighbors, which are the cells that are horizontally, vertically or diagonally adjacent. The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed, births and death occur simultaneously and the discrete moment at which this happens is called 'tick'. Each generation is a pure function of the preceding one. The rules are continuously applied repeatedly to create further generations.

1.2 MOTIVATION:

The Game of Life is not typically a computer game, but a cellular automation. It consists of a collection of cells, which based on the environment around them can live or die or multiply. Depending on the initial conditions, the cells form various patterns throughout its course

Amazed by what four simple rules surprisingly lead to complex, organic patterns, we are excited to simulate this using the concepts we've learnt in OpenGL.

1.3 SCOPE:

- There has been much recent interest in cellular automata, a field of mathematical research. Complicated, persistent structures like pent decathlons and switch engines can evolve naturally from random starting conditions
- Game of Life in Triangular Tessellations is used in ***Image recovery.***

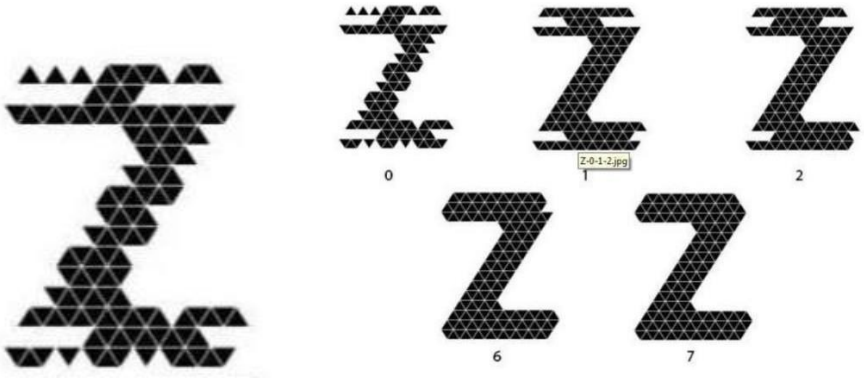


Fig 1.1 Image Recovery

- ***Sequential Fault Convergence:***

  In hardware implementation of CA, the experimental result shows that CA produces better sequential fault convergence than the linear feedback shift register by applying the linear hybrid cellular automata rules.

- ***Memorizing Capacity:***

  The memorizing capacity of a hybrid 3-neighborhood CA is better than that of Hopfield network. The Hopfield network is the model of neural network known for its association capacity.

- ***Simulation Performance:***

  A cellular automata machine can achieve simulation performance of at least several orders of magnitude higher than that can be achieved with a conventional computer at compactable cost.

- ***Theoretical Framework:***

  A theoretical framework to study CA evolution based on graph theoretic formulation a graph named as RVG (Rule Vector Graph) can be derived from the rule vector of a CA employing linear and non-linear rules. CA evolution can be characterized from the study of RVG properties.

- ***Soft Computing:***

  A soft computing tool for CA synthesis a methodology is under development for evolution of SOCA (Self Organizing CA) to realize a given global behavior.

- ***Modelling Tools:***

  Modelling tools based on the CA theory developed, a general methodology is under development to build a CA based model to simulate a system. The modelling tool enables design of a program to be executed on PCA (Programmable CA) to simulate the given system environment.

- ***Pattern Recognition:***

  Pattern recognition in the current Cyber Age, has got wide varieties of applications. CA based Pattern Classification or Clustering methodologies are under development based on the theoretical frame work

- ***CA-Encompression:***

  CA-Encompression (Encryption + Compression), in the current cyber age, large volume of different classes of data - text, image, graphics, video, audio, voice, custom data files -are stored and/or transferred over communication links. Compression and security of such data files are of major concern. Solutions to these problems lie in the development of high speed low cost software/hardware for data compression and data encryption. CA-Encompression technology is being developed as a single integrated operation for both compression and encryption of specific classes of data files such as medical image, voice data, video conference, DNA sequence, Protein sequence etc. Both loss and lossless encompression are under development based on CA model.

- ***CA Compression:***

Standalone CA Compression or CA-encryption technology instead of a single integrated operation of compression and encryption, if a user demands only compression or only encryption, it can be supported using standalone packages (software/hardware version)

- ***CA Based AES:***

  CA based AES (Advanced Encryption System), as AES is the most popular security package, CA based implementation of AES algorithm in underway for development of low cost, high speed hardwired version of AES, is under development.

1.4 PROBLEM STATEMENT:

The existing system follows the following rules:

• Any live cell with fewer than two live neighbors dies, as if caused by under population.

• Any live cell with two or three live neighbors' lives on to the next generation.

• Any live cell with more than three live neighbors dies, as if by over population

• Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

The concept is to show how simple rules can generate complex objects. This is accomplished through an animation created by applying these rules iteratively

1.5 LIMITATIONS:

The existing implementations are restricted to the grid world, and since it is a zero-player game, it solely depends on the initial conditions. This is not suitable, when using this system to mimic populations in real world. Also, all cells are considered same, that is there is no differentiation among cells where as in real world there are difference like age. The pixels are also restricted to black (dead) and white (alive).

1.6 PROPOSED SYSTEM:

We propose to implement and extend this concept, by first adding different kinds of cells that have some additional features. Then we will extend the given rules, by adding new rules, and observe their effects. To make it interactive, we intend to allow the user to add/delete cells during the evolution process.

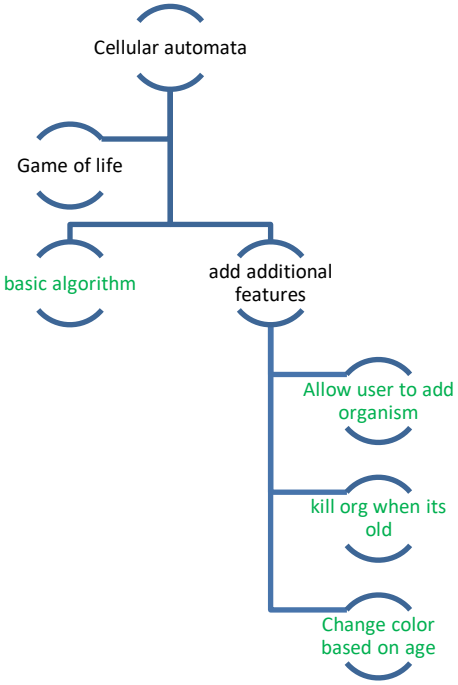


Fig 1.2 Proposed system (Green->Features added)

CHAPTER 2

LITERATURE SURVEY

Von Neumann and Ulam, first proposed the concept of Cellular Automata. The simple structure of CA has attracted researchers from various disciplines. Its application had been proposed in different branches of science. A large number of research papers are published every year. Specialized International Conference and special issues of various journals on CA have been initiated in the last decades. Now several universities have also started offering course on cellular automata.

The popularity of CA can be traced to their simplicity and it is used for the modelling of complex system using simple rule. Cellular automata can be viewed as a simple model of a spatially extended decentralized system made up of a number of individual component is called cell and each individual cell communicates between themselves using the simple rules. Each individual cell is in a specific state which changes over time depending on the state of its local neighbors. The overall structure can be viewed as parallel processing devices.

Cellular automata are a collection of cells that each adapts one of a finite number of states. Single cells change in states by following a local rule that depends on the environment of the cell. The environment of a cell is usually taken to be a small number of neighboring cells, two typical neighborhood option

(a) Von Neumann Neighborhood
(b) Moore Neighborhood.

A CA consists of a regular uniform n-dimensional lattice (or array). At each site of the lattice (cell), a physical quantity takes values. This physical quantity is the global state of the CA, and the value of this quantity at each cell is its local state. Each cell is restricted to the local neighborhood interactions only, and consequently it is incapable of immediate global communication. The neighborhood of the cell is taken to be the cell itself and some or all of the immediately adjacent cells.

A cellular automaton evolves in discrete steps, with the next value of one site determined by its previous value and that of a set of sites called the neighbor sites. The extent of the neighborhood can vary, depending among other factors upon the dimensionality of the cellular automaton under consideration. The width of the neighborhood of the cell j is the width of the neighborhood at the jth side of the array.

The state of each cell is updated simultaneously at discrete time steps, based on the states in its neighborhood at the preceding time step. The algorithm used to compute the next cell state is referred to as the CA local rule.

Typically, a cellular automaton consists of a graph where each node is a finite state automaton (FSA) or cell. This graph is usually in the form of a two-dimensional lattice whose cells evolve according to a global update function applied uniformly over all the cells. As arguments, this update function takes the cells present state and the states of the cells in its interaction neighborhood.

CA is a bio-inspired paradigm highly addressing the soft computing and hardware for large class of applications including information security. They are a particular class of dynamical systems that enable to describe the evolution of complex systems with simple rules, without using partial differential equations. Typically, a cellular automaton consists of graph where each node is a Finite State Automaton (FSA) or cell. This graph is usually in the form of a two-dimensional lattice whose cells evolve according to a global update function applied uniformly over all the cells.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 HARDWARE REQUIREMENTS:

- Processor :Core i5/i7
- RAM: :1 GB
- Hard Disk :2 GB
- Monitor :LCD/LED Monitor
- Keyboard :Normal/Multimedia

3.2 SOFTWARE REQUIREMENTS:

- Operating System :Windows 10
- Code blocks software
- OpenGL (Open Graphics Library)
- Built-in graphics : GLUT

Language: C

CHAPTER 4

SYSTEM ANALYSIS

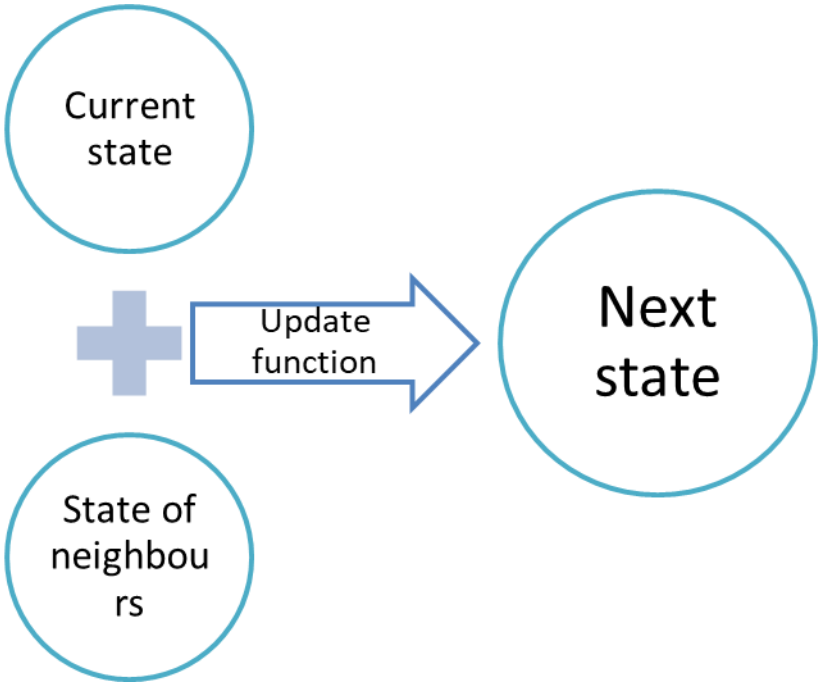


Fig 4.1. State transition depends on neighboring state

Neighbor 1	Neighbor 2	Neighbor 3
Neighbor 4	Cell of interest	Neighbor 5
Neighbor 6	Neighbor 7	Neighbor 8

Fig 4.2. How Neighbors of a cell are calculated

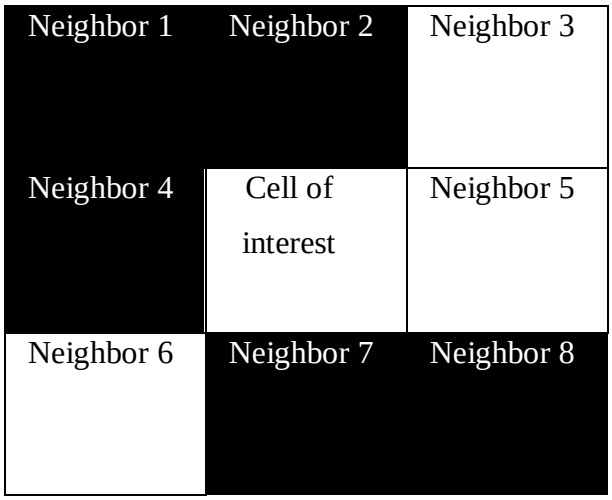


Fig 4.3. Cell with 3 neighbors

(White- Alive, Black- Dead)

Here in the next iteration Neighbor 2, Neighbor 7 becomes alive (as they have exactly 3 neighbors)

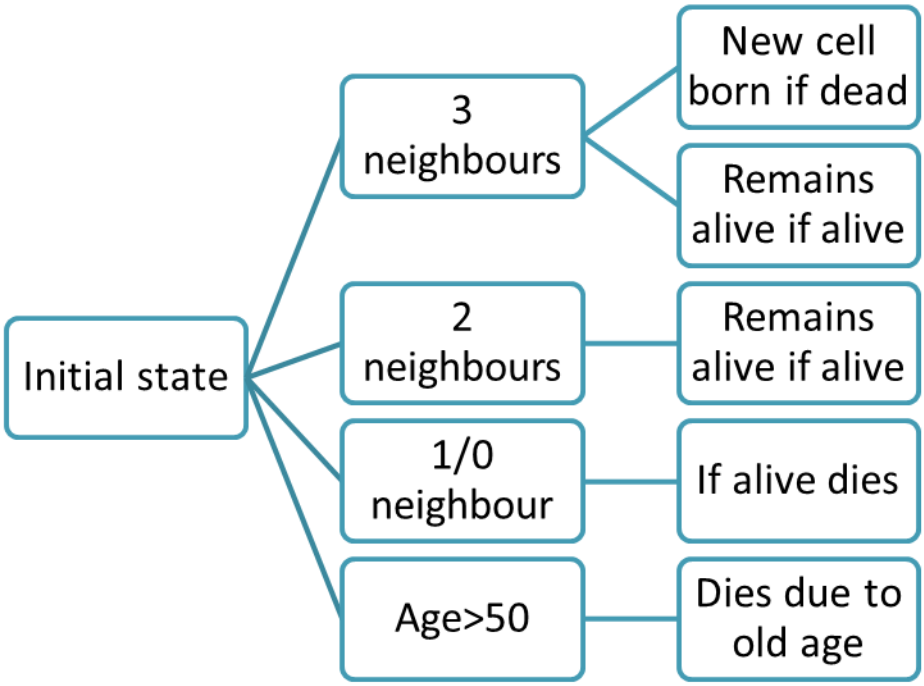


Fig 4.4. Game of Life

CHAPTER 5

IMPLEMENTATION

5.1 FEATURES IMPLEMENTED:

- Initialize random state
- Automatically run next iteration (using timer function)
- Add concept of old age, as a new rule that can be seen by changing colors
    - The maximum age that an organism can be alive is also set, after which it dies.
- Animation (using double buffering mode)
    - To increase/decrease animation speed the time parameter can be modified
- User interactivity (using keyboard and mouse)
    - Keyboard:
        - Press 1 to play animation
        - Press 2 to pause
        - Press 3 to reset board to blank state
    - Mouse:
        - Left click/ drag to add pixels on screen (in pause mode)

This project is designed using some of the OpenGL inbuilt functions and some user defined functions. This chapter comprises of the complete explanation of the design of the project using various OpenGL functions and user defined methods.

5.2 OPENGL BUILT-IN FUNCTIONS:

The different OpenGL functions that are used in the project is described below:

1) glClearColor() :

glClearColor() specifies the red, green, blue, and alpha values used when the color buffers are cleared. Values specified by glClearColor() are clamped to the range 0, 1

2) glMatrixMode(GL_PROJECTION):

glMatrixMode specifies which matrix stack is the target for subsequent matrix operations. It sets the current matrix mode. Mode can take one of GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE, GL_COLOR values. glMatrixMode(GL_PROJECTION) applies the subsequent matrix operations to the projection matrix stack.

3) glLoadIdentity():

glLoadIdentity replaces the current matrix with the identity matrix. It is semantically equivalent to calling glLoadMatrix() with the identity matrix. This function has no parameters and does not return a value.

4) glOrtho():

glOrtho() multiplies the current matrix with an orthographic matrix. It describes a transformation that produces a parallel projection. The current matrix is multiplied by this orthographic matrix and replaces the current matrix. It's parameters are left, right (Specify the coordinates for the left and right clipping planes). Bottom, top (Specifies the coordinates for the bottom and top horizontal clipping planes. Near, far (Specify the distances to the nearer and farther depth clipping planes.

5) glClear(GL_COLOR_BUFFER_BIT):

glClear() clears the buffers to preset values. Bitwise OR of masks that indicate the buffers to be cleared. The four masks are GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT, GL_ACCUM_BUFFER_BIT and GL_STENCIL_BUFFER_BIT.

6) glPointSize(point_size):

glPointSize() specifies the diameter of the rasterized points. If point size mode is disabled, this value will be used to rasterize points. Otherwise the values written to the shading language built-in variable gl_PointSize will be used.

7) glFlush():

Force execution of GL commands in finite time. glFlush empties all buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time. glFlush can return at any time. It does not wait until the execution of all previously issued GL commands is complete.

8) glBegin(GLenum mode) & glEnd():

Delimits the vertices of a primitive or a group of like primitives. It specifies the primitive/primitives that will be created from vertices presented between glBegin() and the subsequent glEnd(). Ten symbolic constants are: GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS, GL_QUAD_STRIP, GL_POLYGON. glBegin(GL_POINTS) treats each vertex as a single point n. N points are drawn.

9) glVertex2f(GLfloat x, GLfloat y):

Specifies a vertex. glVertex commands are used within glBegin/g lEnd pairs to specify point, line, and polygon vertices. The current color, normal, texture coordinates, and fog coordinate are associated with the vertex when glVertex is called.

10) glutSwapBuffers() :

glutSwapBuffers() swaps the buffers of the current window if double buffered. It performs a buffer swap on the layer in use for the current window. Specifically, glutSwapBuffers() promotes the contents of the back buffer of the layer in use of the current window to become the contents of the front buffer. An implicit glFlush() is done by glutSwapBuffers() before it returns. If the layer in use is not double buffered, glutSwapBuffers has no effect.

11) glutInit(&argc,argv):

glutInit() is used to initialize the GLUT library. glutInit() will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit() may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. glutInit() also processes command line options, but the specific options parse are window system dependent. It takes two parameters '&argc' – A pointer to the program's unmodifies argc variable from main. Upon return, the value spointed to by argc pointer will be updated, because glutInit() extracts any command line options intended for the GLUT library. 'argv' – The program's unmodified argv variable from main. Like argcp, the data for argv will be updated because glutInit() extracts command line options understood by the GLUT library.

12) glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB):

It sets the initial display mode. The initial display mode is used when creating top-level windows, sub windows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay. GLUT_DOUBLE is a bit mask to select a double buffered window. GLUT_RGB is a bit mask to select an RGB mode window.

13) glutCreateWindow():

It creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name. Implicitly, the current window is set to the newly created window.

14) glutInitWindowSize():

This sets the initial window size. Windows created by glutCreateWindow() will be requested to be created with the current initial window position and size. The initial value of the initial window size GLUT state is 300 by 300. The initial window size components must be greater than zero. The intent of the initial window position and size values is to provide a suggestion to the window system for a window's initial size and position. The window system is not obligated to use this information. A GLUT program should use the window's reshape callback to determine the true size of the window.

15) glutInitWindowPosition():

It sets the initial window position. The window system is not obligated to use this information. Hence, GLUT programs should not assume the window was created at the specified size or position. A GLUT program should use the windows reshape callback to determine the true size of the window.

16) GetAsyncKeyState(vKey):

It determines whether a key is up or down at the time the function is called, and whether the key was pressed after a previous call to GetAsyncKeyState(). It has a parameter vKey of type 'int' and a return value of type 'short'. If the function succeeds, the return value specifies whether the key was pressed since the last call to GetAsyncKetState() and whether the key is currently up or down. If the most significant bit is set, the key is down, and if the least significant bit is set, the key was pressed after the previous call to GetAsyncKeyState().

17) glutDisplayFunc(display):

It registers the callback function (or event handler) for handling window-paint event. The OpenGL graphic system calls back this handler when it receives a window re-paint request. In the example, we register the function display() as the handler.

18) glutKeyboardFunc():

glutKeyboardFunc() sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The state of

modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data.

19) glutMouseFunc(mouse):

glutMouseFunc(mouse) sets the mouse callback for the current window. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse call back. The button parameter can be one of GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON and GLUT_MIDDLE_BUTTON. The state parameter is either GLUT_UP or GLUT_DOWN indicating whether a callback was due to release or press respectively.

20) glutMainLoop():

glutMainLoop() enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

21) glutMotionFunc():

glutMotionFun() set the motion and passive motion callbacks respectively for the current window. The motion callback for a window is called when the mouse within the window while one or more mouse buttons are pressed. It has two parameters x snd y that indicate the mouse location in window relative co-ordinates.

22) glutTimerFunc():

glutTimerFunc() registers a timer callback to be triggered in a specified number of milli seconds. The value parameter to the timer callback will be the value of the value parameter to glutTimerFunc(). The number of milli seconds is a lower bound on the time before the callback is generated. GLUT attempts to deliver the timer callback as soon as possible after the expiration of the callback's time interval.

5.3 USER DEFINEED FUNCTIONS:

1) Init():

Used to initialize the display window. The background color is set as black and Identity matrix is loaded. The Matrix mode is set to Projection, so that the animations are applied. Orthographic projection is use, as it is a flat 2D visualization.

2) Struct P:

The organism is created using this structure. Each organism is initialized with 3 attributes. They are:

- Alive – Boolean value that records if the organism in that cell is alive or dead
- Age – tracks the age of the organism, initialized with 0. Can take values up to 50, after which organism dies and age is reset to 0.
- Next – tracks the state of the organism during the next iteration. Helps in animation

3) Display():

Used to display the board with the organism. Iterates through all cells on the board and draws the organism if it is alive. The color of the organism is set here according to its age. If user adds organisms with mouse, it is drawn using the mouse coordinates here. As double buffer mode is use, we call glutswapbuffers to display the final image.

4) Death():

This function decides the next state, that is it sets which organism will be born, alive, dead for every iteration. This is carried out using the rules in Conway's game of life. It used the concept of neighbors. First the number of alive neighbors for the current cell is calculated. Then if the current cell is alive, we check for the following 2 conditions:

- If alive neighbors are less than 2 or greater than 3, the cell dies in the next state
- If alive neighbors are exactly 2 or 3, the cell remains alive for the next iteration

If the current cell is dead, and has exactly 3 alive neighbors, it become alive in the next state.

If the age of the current cell exceeds 50, then it dies in the next iteration.

5) Timer():

This function is used to call the display function repeatedly every 0.1s, to have a smooth animation. It also has options to pause, play, reset the game. This is achieved by using the function GetAsynchKeyState() that monitors the key pressed on the keyboard.

- If 1 is pressed, the game runs (default mode)
- If 2 is pressed the game enters pause state, where the animation stops, and the user can add organisms.
- If 3 is pressed the board is reset. This is achieved by setting all cells as dead

In play mode (1) the death() function is called to get the next state, followed by display function.

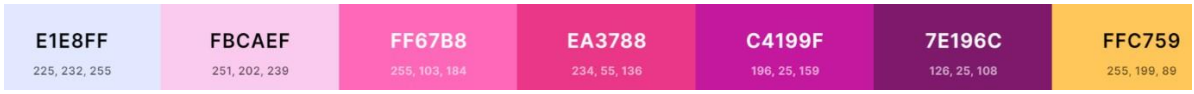
6) Mouse():

This function is used to track the mouse position and state. This is called by glutMouseFunc()

7) Motion():

Used to track the positions of mouse is moved on screen. This is called by glutMotionFunc() and is used to add organisms when user clicks and drags the mouse on screen.

5.4 COLOR SCHEME USED:



age<5	age <10	age <15	age <20	age <25	age <45	age <50

Fig 5.4 Color Scheme

CHAPTER 6

VISUALIZATION OF RESULTS

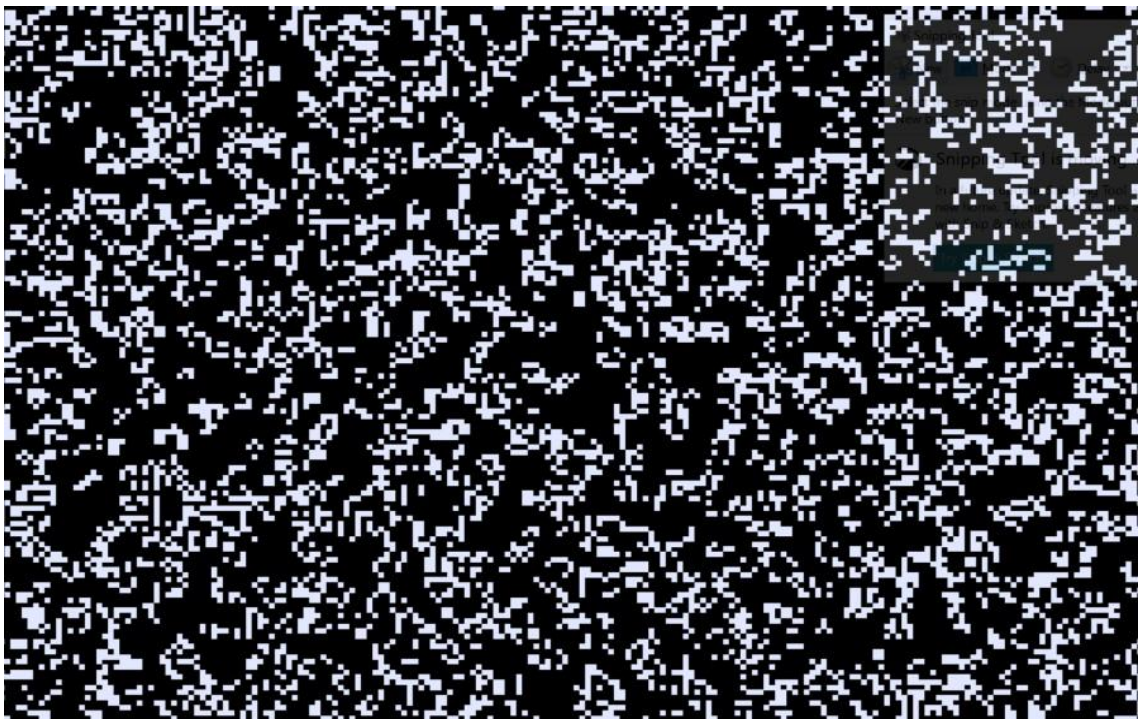


Fig 6.1. Randomly generated initial state

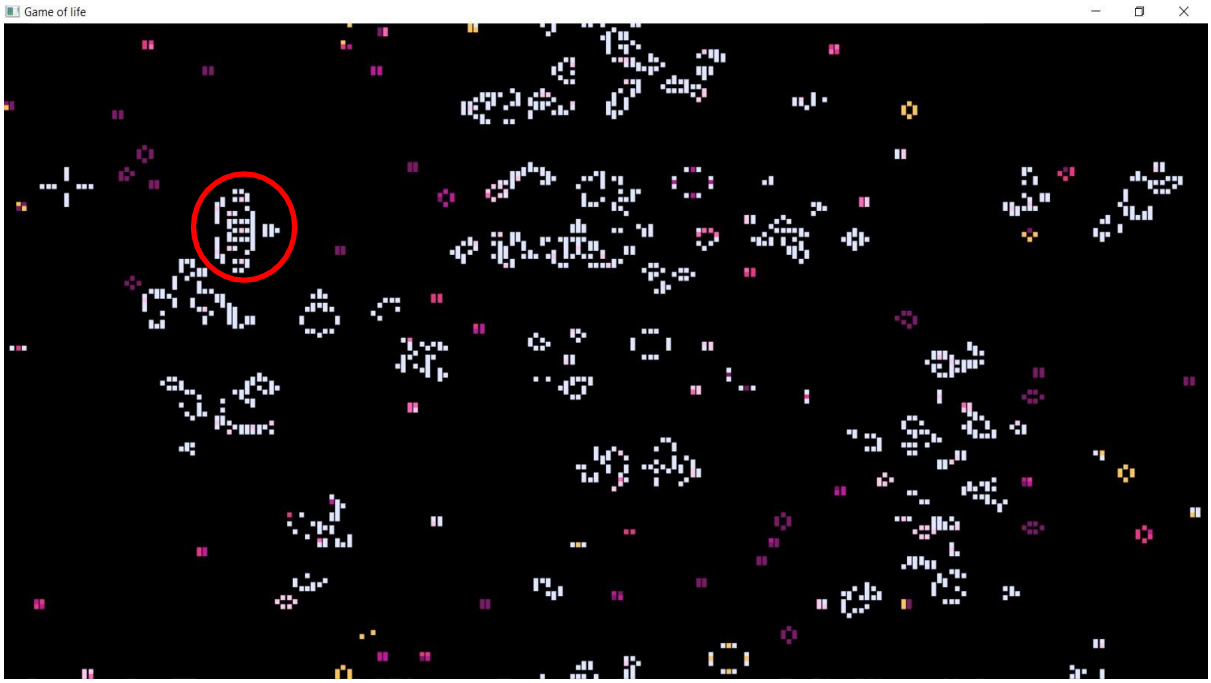


Fig 6.2 State after few iterations (observe formation of patterns, highlighted in red)

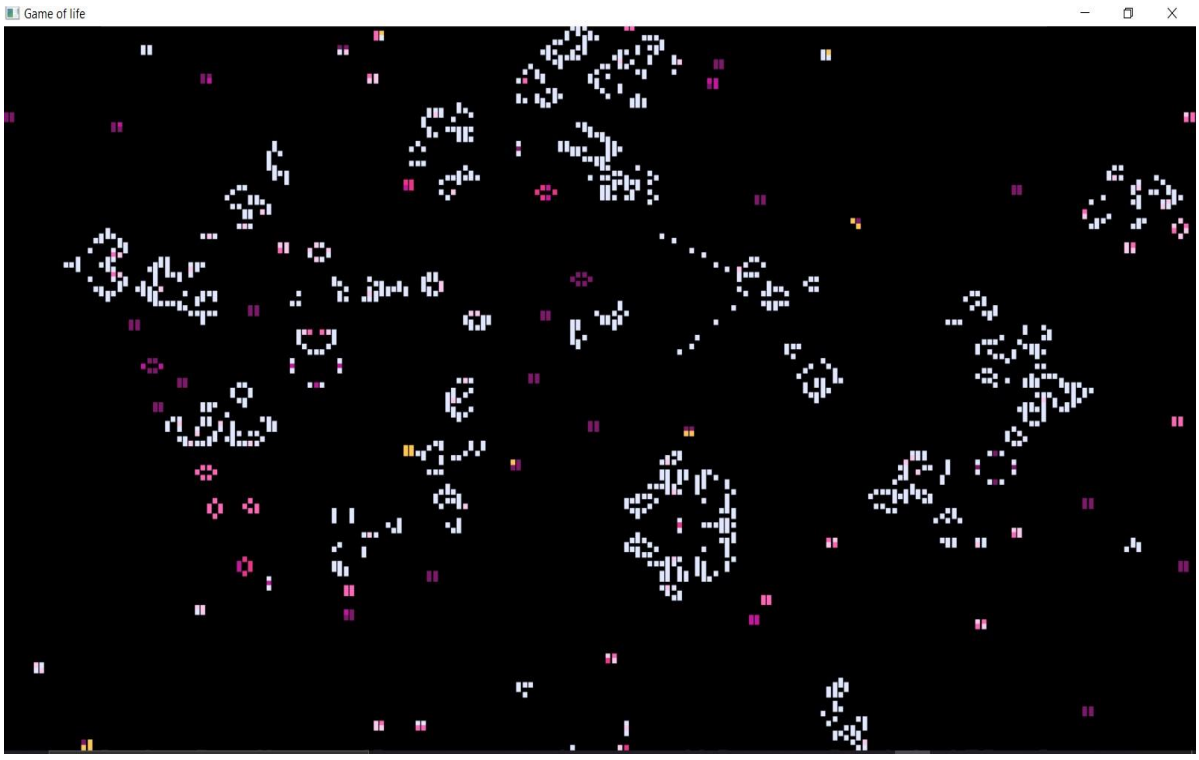


Fig 6.3 Pause mode showing user adding pixels

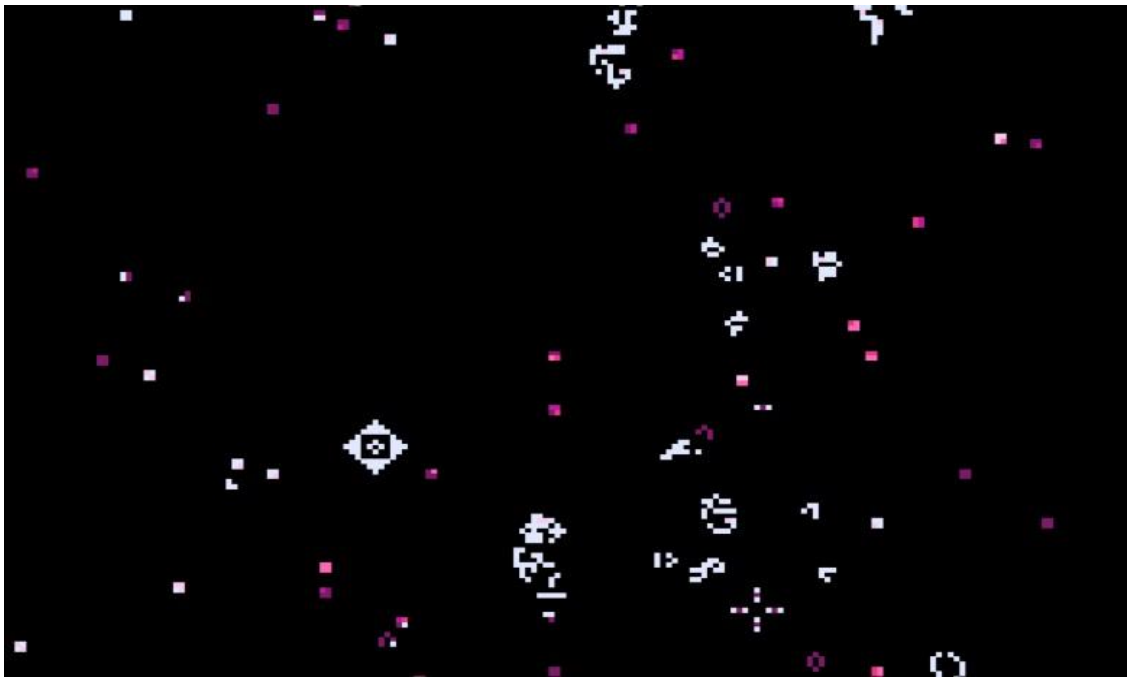


Fig 6.4 Near End State

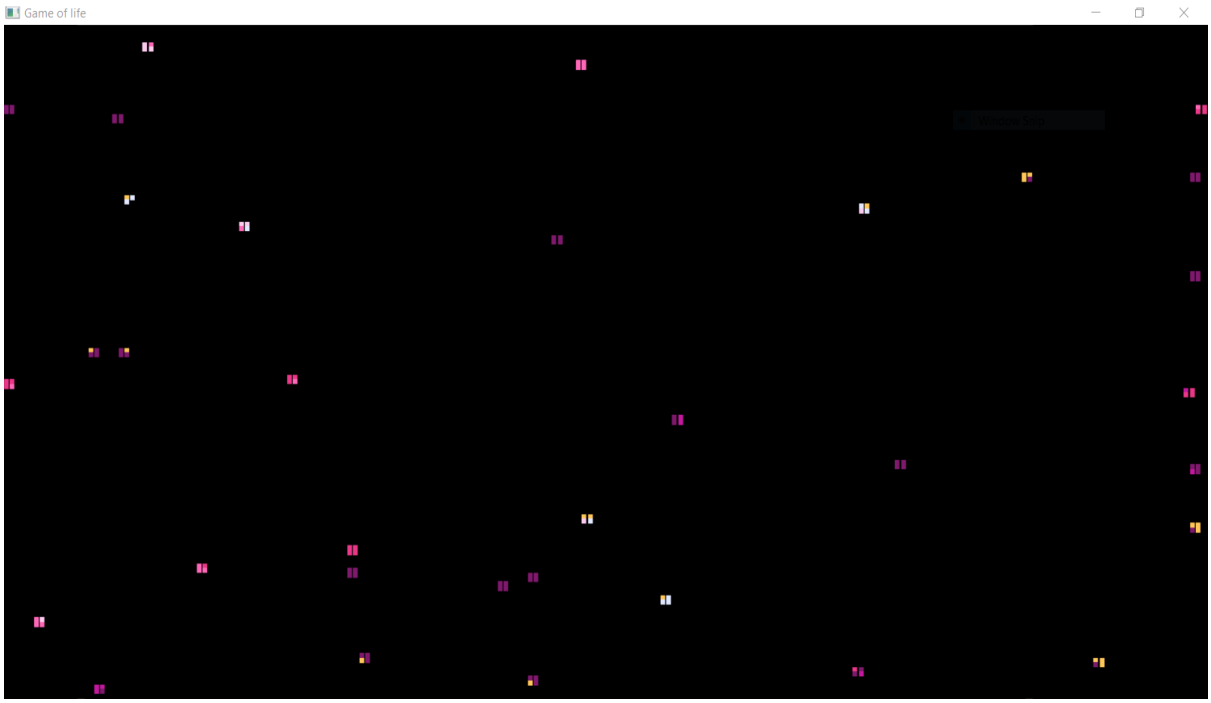


Fig 6.5 Final state, successive iterations cause only change in age, the number of births and deaths are equal.

CONCLUSION

Through this project we have implemented our version of Conway's game of life. On analyzing the resulting animations, we observe the following:

- The addition of color makes it visually appetizing in the animation
- Due to the concept of old age, we observe that for lower maximum ages, the complexity of patterns created decreases, as the pixels automatically die.
- We also observe formation of colonies. That is clusters of pixels are observed in random locations on screen. These do not die, but instead keep on looping in a cycle
- The colonies can be destroyed by certain moving pixels, that resemble bullets from a gun, these destroy the colony on contact
- As the animation progresses, we observe a stable state being achieved. The number of new cells born is equivalent to number of deaths. Here death is caused only due to old age. There is no movement of pixels.

Thus, we observe the progression of a chaotic initial state, to a stable final state.

FUTURE ENHANCEMENTS

Few ideas for the future iterations of this project

- The feature to delete existing organisms
- Changing type of organism, that is from pixel to plants or animals

REFERENCES

- https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
- https://www.cse.msu.edu/~cse872/tutorial4.html
- https://kylewbanks.com/blog/tutorial-opengl-with-golang-part-1-hello-opengl
- https://people.sc.fsu.edu/~jburkardt/c_src/life_opengl/life_opengl.c
- https://www.opengl.org/resources/libraries/glut/spec3/node64.html
- https://www.opengl.org/resources/libraries/glut/spec3/node50.html
- https://www.opengl.org/resources/libraries/glut/spec3/node49.html