Conway's Game of Life

1.0

Generated by Doxygen 1.9.8

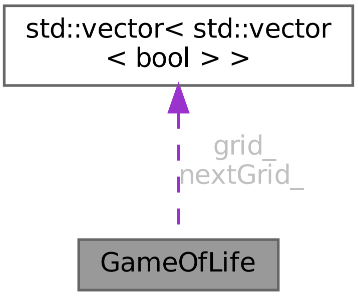
1 Class Documentation

1.1 GameOfLife Class Reference

Class implementing Conway's Game of Life simulation.

```
#include <GameOfLife.h>
```

Collaboration diagram for GameOfLife:



Public Member Functions

- GameOfLife (int height, int width)

  Constructor for the GameOfLife class.
- ∼GameOfLife ()

  Destructor for the GameOfLife class.
- void initializeRandom ()

  Initialize the game with a random pattern.
- void initializePattern (const std::vector< std::vector< bool > > &pattern)

  Initialize the game with a specific pattern.
- void run ()

  Run the game simulation.
- void update ()

  Update the game state for one generation.
- void draw ()

  Draw the current game state.
- bool saveAsBMP (const std::string &filename)

  Save the current game state as a BMP image.

Private Member Functions

- void initNCurses ()

  Initialize ncurses settings.
- int countNeighbors (int row, int col)

  Count the number of live neighbors for a cell.
- std::string generateTimestamp ()

  Generate a timestamp string for filenames.
- void writeBMPHeader (std::ofstream &file, int width, int height)

  Write BMP file header.
- void writeBMPData (std::ofstream &file)

  Write BMP image data.

Private Attributes

- int height_

  Height of the game grid.
- int width_

  Width of the game grid.
- std::vector< std::vector< bool > > grid_

  Current state of the game grid.
- std::vector< std::vector< bool > > nextGrid_

  Next state of the game grid.
- bool running_

  Flag indicating if the game is running.
- int generation_

  Current generation count.

1.1.1 Detailed Description

Class implementing Conway's Game of Life simulation.

This class provides functionality to run and visualize Conway's Game of Life using the ncurses library for terminal-based visualization.

Definition at line 26 of file GameOfLife.h.

1.1.2 Constructor & Destructor Documentation

GameOfLife()

```
GameOfLife::GameOfLife (
              int height,
              int width )
```

Constructor for the GameOfLife class.

Parameters

height	The height of the game grid
width	The width of the game grid

Definition at line 8 of file GameOfLife.cpp.

```
00009     : height_(height), width_(width), running_(true), generation_(0) {
00010     // Initialize grid with all cells dead
00011     grid_.resize(height_, std::vector<bool>(width_, false));
00012     nextGrid_.resize(height_, std::vector<bool>(width_, false));
00013 
00014     // Initialize ncurses
00015     initNCurses();
00016 
00017     // Seed random number generator
00018     std::srand(std::time(nullptr));
00019 }
```

∼GameOfLife()

```
GameOfLife::∼GameOfLife ( )
```

Destructor for the GameOfLife class.

Definition at line 21 of file GameOfLife.cpp.

```
00021                             {
00022     // End ncurses mode
00023     endwin();
00024 }
```

1.1.3 Member Function Documentation

countNeighbors()

```
int GameOfLife::countNeighbors (
              int row,
              int col )  [private]
```

Count the number of live neighbors for a cell.

Parameters

row	The row of the cell
col	The column of the cell

Returns

The number of live neighbors

Definition at line 158 of file GameOfLife.cpp.

```
00158                                                     {
00159     int count = 0;
00160 
00161     // Check all 8 neighboring cells
00162     for (int i = -1; i <= 1; i++) {
00163         for (int j = -1; j <= 1; j++) {
00164             // Skip the cell itself
00165             if (i == 0 && j == 0) continue;
00166 
00167             // Calculate neighbor coordinates with wrapping
00168             int neighborRow = (row + i + height_) % height_;
00169             int neighborCol = (col + j + width_) % width_;
00170 
00171             // Count live neighbors
00172             if (grid_[neighborRow][neighborCol]) {
00173                 count++;
00174             }
00175         }
00176     }
00177 
00178     return count;
00179 }
```

draw()

```
void GameOfLife::draw ( )
```

Draw the current game state.

Definition at line 181 of file GameOfLife.cpp.

```
00181                            {
00182     for (int i = 0; i < height_; i++) {
00183         for (int j = 0; j < width_; j++) {
00184             if (grid_[i][j]) {
00185                 // Cell is alive
00186                 if (has_colors()) {
00187                     attron(COLOR_PAIR(1));
00188                     mvaddch(i, j, ' ');
00189                     attroff(COLOR_PAIR(1));
00190                 } else {
00191                     mvaddch(i, j, '#');
00192                 }
00193             } else {
00194                 // Cell is dead
00195                 if (has_colors()) {
00196                     attron(COLOR_PAIR(2));
00197                     mvaddch(i, j, ' ');
00198                     attroff(COLOR_PAIR(2));
00199                 } else {
00200                     mvaddch(i, j, '.');
00201                 }
00202             }
00203         }
00204     }
00205 
00206     // Update the screen
00207     refresh();
00208 }
```

generateTimestamp()

```
std::string GameOfLife::generateTimestamp ( )  [private]
```

Generate a timestamp string for filenames.

Returns

A string containing the current timestamp

Definition at line 210 of file GameOfLife.cpp.

```
00210                                          {
00211     auto now = std::time(nullptr);
00212     auto tm = *std::localtime(&now);
00213 
00214     std::ostringstream oss;
00215     oss « std::put_time(&tm, "%Y%m%d_%H%M%S");
00216     return oss.str();
00217 }
```

initializePattern()

```
void GameOfLife::initializePattern (
               const std::vector< std::vector< bool > > & pattern )
```

Initialize the game with a specific pattern.

Parameters

pattern	The pattern to initialize with

Definition at line 52 of file GameOfLife.cpp.

```
00052                                                                              {
00053     int patternHeight = pattern.size();
00054     int patternWidth = pattern[0].size();
00055 
00056     // Calculate starting position to center the pattern
00057     int startRow = (height_ - patternHeight) / 2;
00058     int startCol = (width_ - patternWidth) / 2;
00059 
00060     // Clear the grid first
00061     for (int i = 0; i < height_; i++) {
00062         for (int j = 0; j < width_; j++) {
00063             grid_[i][j] = false;
00064         }
00065     }
00066 
00067     // Place the pattern in the center
00068     for (int i = 0; i < patternHeight; i++) {
00069         for (int j = 0; j < patternWidth; j++) {
00070             int row = startRow + i;
00071             int col = startCol + j;
00072             if (row >= 0 && row < height_ && col >= 0 && col < width_) {
00073                 grid_[row][col] = pattern[i][j];
00074             }
00075         }
00076     }
00077 }
```

initializeRandom()

```
void GameOfLife::initializeRandom ( )
```

Initialize the game with a random pattern.

Definition at line 43 of file GameOfLife.cpp.

```
00043                                    {
00044     for (int i = 0; i < height_; i++) {
00045         for (int j = 0; j < width_; j++) {
00046             // 25% chance of a cell being alive
00047             grid_[i][j] = (std::rand() % 4 == 0);
00048         }
00049     }
00050 }
```

initNCurses()

`void GameOfLife::initNCurses ( )  [private]`

Initialize ncurses settings.

Definition at line 26 of file GameOfLife.cpp.

```
00026                               {
00027     // Initialize ncurses
00028     initscr();
00029     cbreak();
00030     noecho();
00031     keypad(stdscr, TRUE);
00032     curs_set(0);  // Hide cursor
00033     timeout(100); // Set getch() non-blocking with 100ms timeout
00034 
00035     // Check if terminal supports colors
00036     if (has_colors()) {
00037         start_color();
00038         init_pair(1, COLOR_BLACK, COLOR_GREEN);  // Live cells
00039         init_pair(2, COLOR_BLACK, COLOR_BLACK);  // Dead cells
00040     }
00041 }
```

run()

`void GameOfLife::run ( )`

Run the game simulation.

Definition at line 79 of file GameOfLife.cpp.

```
00079                       {
00080     while (running_) {
00081         // Clear screen
00082         clear();
00083 
00084         // Draw the current state
00085         draw();
00086 
00087         // Display generation count
00088         mvprintw(height_ + 1, 0, "Generation: %d", generation_);
00089         mvprintw(height_ + 2, 0, "Press ’q’ to quit, ’s’ to save image, ’r’ to randomize");
00090 
00091         // Process input
00092         int ch = getch();
00093         switch (ch) {
00094             case ’q’:
00095             case ’Q’:
00096                 running_ = false;
00097                 break;
00098             case ’s’:
00099             case ’S’:
00100                 {
00101                     std::string filename = "gameoflife_" + generateTimestamp() + ".bmp";
00102                     if (saveAsBMP(filename)) {
00103                         mvprintw(height_ + 3, 0, "Image saved as %s", filename.c_str());
00104                         refresh();
00105                         std::this_thread::sleep_for(std::chrono::milliseconds(1000));
00106                     } else {
00107                         mvprintw(height_ + 3, 0, "Failed to save image");
00108                         refresh();
00109                         std::this_thread::sleep_for(std::chrono::milliseconds(1000));
00110                     }
00111                 }
00112                 break;
00113             case 'r':
00114             case 'R':
00115                 initializeRandom();
00116                 generation_ = 0;
00117                 break;
00118         }
00119 
00120         // Update the game state
00121         update();
00122         generation_++;
00123 
00124         // Small delay to control game speed
00125         std::this_thread::sleep_for(std::chrono::milliseconds(100));
00126     }
00127 }
```

saveAsBMP()

```
bool GameOfLife::saveAsBMP (
              const std::string & filename )
```

Save the current game state as a BMP image.

Parameters

filename	The name of the file to save to

Returns

true if the save was successful, false otherwise

Definition at line 219 of file GameOfLife.cpp.

```
00219                                                     {
00220     std::ofstream file(filename, std::ios::binary);
00221     if (!file) {
00222         return false;
00223     }
00224 
00225     // Write BMP header
00226     writeBMPHeader(file, width_, height_);
00227 
00228     // Write BMP data
00229     writeBMPData(file);
00230 
00231     file.close();
00232     return true;
00233 }
```

update()

```
void GameOfLife::update ( )
```

Update the game state for one generation.

Definition at line 129 of file GameOfLife.cpp.

```
00129                         {
00130     // Calculate the next generation
00131     for (int i = 0; i < height_; i++) {
00132         for (int j = 0; j < width_; j++) {
00133             int neighbors = countNeighbors(i, j);
00134 
00135             // Apply Conway's Game of Life rules
00136             if (grid_[i][j]) {
00137                 // Cell is alive
00138                 if (neighbors < 2 || neighbors > 3) {
00139                     nextGrid_[i][j] = false; // Cell dies
00140                 } else {
00141                     nextGrid_[i][j] = true;  // Cell survives
00142                 }
00143             } else {
00144                 // Cell is dead
00145                 if (neighbors == 3) {
00146                     nextGrid_[i][j] = true;  // Cell becomes alive
00147                 } else {
00148                     nextGrid_[i][j] = false; // Cell stays dead
00149                 }
00150             }
00151         }
00152     }
00153 
00154     // Update the grid with the new generation
00155     grid_ = nextGrid_;
00156 }
```

writeBMPData()

```
void GameOfLife::writeBMPData (
               std::ofstream & file )  [private]
```

Write BMP image data.

Parameters

file	The file to write to

Definition at line 287 of file GameOfLife.cpp.

```
00287                                                 {
00288     // Calculate row size and padding
00289     int rowSize = ((width_ * 24 + 31) / 32) * 4;
00290     int paddingSize = rowSize - width_ * 3;
00291     unsigned char padding[3] = {0, 0, 0};
00292 
00293     // BMP stores images bottom-up
00294     for (int i = height_ - 1; i >= 0; i--) {
00295         for (int j = 0; j < width_; j++) {
00296             unsigned char color[3];
00297 
00298             // Set pixel color (BGR format)
00299             if (grid_[i][j]) {
00300                 // Live cell - green
00301                 color[0] = 0;    // Blue
00302                 color[1] = 255;  // Green
00303                 color[2] = 0;    // Red
00304             } else {
00305                 // Dead cell - black
00306                 color[0] = 0;    // Blue
00307                 color[1] = 0;    // Green
00308                 color[2] = 0;    // Red
00309             }
00310 
00311             // Write pixel data
00312             file.write(reinterpret_cast<char*>(color), 3);
00313         }
00314 
00315         // Write padding
00316         if (paddingSize > 0) {
00317             file.write(reinterpret_cast<char*>(padding), paddingSize);
00318         }
00319     }
00320 }
```

writeBMPHeader()

```
void GameOfLife::writeBMPHeader (
               std::ofstream & file,
               int width,
               int height )  [private]
```

Write BMP file header.

Parameters

file	The file to write to
width	The width of the image
height	The height of the image

Definition at line 235 of file GameOfLife.cpp.

```
00235                                                                               {
00236     // Calculate row size and padding
00237     int rowSize = ((width * 24 + 31) / 32) * 4;
00238     int paddingSize = rowSize - width * 3;
00239     int fileSize = 54 + rowSize * height;
00240 
00241     // BMP file header (14 bytes)
00242     unsigned char bmpFileHeader[14] = {
00243         ’B’, ’M’,                           // Signature
00244         0, 0, 0, 0,                         // File size in bytes
00245         0, 0, 0, 0,                         // Reserved
00246         54, 0, 0, 0                         // Offset to start of pixel data
00247     };
00248 
00249     // Update file size in header
00250     bmpFileHeader[2] = (unsigned char)(fileSize);
00251     bmpFileHeader[3] = (unsigned char)(fileSize » 8);
00252     bmpFileHeader[4] = (unsigned char)(fileSize » 16);
00253     bmpFileHeader[5] = (unsigned char)(fileSize » 24);
00254 
00255     // BMP info header (40 bytes)
00256     unsigned char bmpInfoHeader[40] = {
00257         40, 0, 0, 0,                        // Info header size
00258         0, 0, 0, 0,                         // Width
00259         0, 0, 0, 0,                         // Height
00260         1, 0,                               // Number of color planes
00261         24, 0,                              // Bits per pixel
00262         0, 0, 0, 0,                         // Compression
00263         0, 0, 0, 0,                         // Image size
00264         0, 0, 0, 0,                         // X pixels per meter
00265         0, 0, 0, 0,                         // Y pixels per meter
00266         0, 0, 0, 0,                         // Colors in color table
00267         0, 0, 0, 0                          // Important color count
00268     };
00269 
00270     // Update width in header
00271     bmpInfoHeader[4] = (unsigned char)(width);
00272     bmpInfoHeader[5] = (unsigned char)(width » 8);
00273     bmpInfoHeader[6] = (unsigned char)(width » 16);
00274     bmpInfoHeader[7] = (unsigned char)(width » 24);
00275 
00276     // Update height in header
00277     bmpInfoHeader[8] = (unsigned char)(height);
00278     bmpInfoHeader[9] = (unsigned char)(height » 8);
00279     bmpInfoHeader[10] = (unsigned char)(height » 16);
00280     bmpInfoHeader[11] = (unsigned char)(height » 24);
00281 
00282     // Write headers
00283     file.write(reinterpret_cast<char*>(bmpFileHeader), 14);
00284     file.write(reinterpret_cast<char*>(bmpInfoHeader), 40);
00285 }
```

1.1.4 Member Data Documentation

generation_

```
int GameOfLife::generation_  [private]
```

Current generation count.

Definition at line 112 of file GameOfLife.h.

grid_

```
std::vector<std::vector<bool> > GameOfLife::grid_  [private]
```

Current state of the game grid.

Definition at line 109 of file GameOfLife.h.

height_

```
int GameOfLife::height_  [private]
```

Height of the game grid.

Definition at line 107 of file GameOfLife.h.

nextGrid_

```
std::vector<std::vector<bool> > GameOfLife::nextGrid_  [private]
```

Next state of the game grid.

Definition at line 110 of file GameOfLife.h.

running_

```
bool GameOfLife::running_  [private]
```

Flag indicating if the game is running.

Definition at line 111 of file GameOfLife.h.

width_

```
int GameOfLife::width_  [private]
```

Width of the game grid.

Definition at line 108 of file GameOfLife.h.

The documentation for this class was generated from the following files:

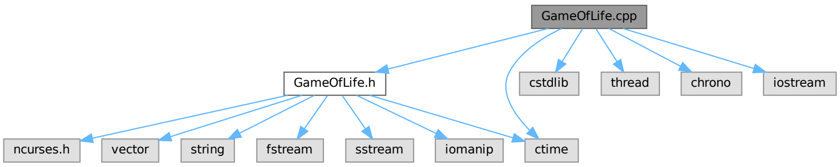
- GameOfLife.h
- GameOfLife.cpp

2 File Documentation

2.1 GameOfLife.cpp File Reference

```
#include "GameOfLife.h"
#include <cstdlib>
#include <ctime>
#include <thread>
#include <chrono>
#include <iostream>
```

Include dependency graph for GameOfLife.cpp:



2.2 GameOfLife.cpp

Go to the documentation of this file.

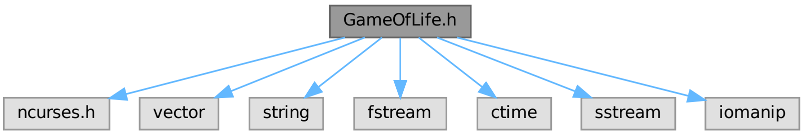
```
#include "GameOfLife.h"
#include <cstdlib>
#include <ctime>
#include <thread>
#include <chrono>
#include <iostream>

GameOfLife::GameOfLife(int height, int width)
    : height_(height), width_(width), running_(true), generation_(0) {
    // Initialize grid with all cells dead
    grid_.resize(height_, std::vector<bool>(width_, false));
    nextGrid_.resize(height_, std::vector<bool>(width_, false));

    // Initialize ncurses
    initNCurses();

    // Seed random number generator
    std::srand(std::time(nullptr));
}

GameOfLife::~GameOfLife() {
    // End ncurses mode
    endwin();
}

void GameOfLife::initNCurses() {
    // Initialize ncurses
    initscr();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);
    curs_set(0);  // Hide cursor
    timeout(100); // Set getch() non-blocking with 100ms timeout

    // Check if terminal supports colors
    if (has_colors()) {
        start_color();
        init_pair(1, COLOR_BLACK, COLOR_GREEN);  // Live cells
        init_pair(2, COLOR_BLACK, COLOR_BLACK);  // Dead cells
    }
}

void GameOfLife::initializeRandom() {
    for (int i = 0; i < height_; i++) {
        for (int j = 0; j < width_; j++) {
            // 25% chance of a cell being alive
            grid_[i][j] = (std::rand() % 4 == 0);
        }
    }
}

void GameOfLife::initializePattern(const std::vector<std::vector<bool»& pattern) {
    int patternHeight = pattern.size();
    int patternWidth = pattern[0].size();

    // Calculate starting position to center the pattern
    int startRow = (height_ - patternHeight) / 2;
    int startCol = (width_ - patternWidth) / 2;

    // Clear the grid first
    for (int i = 0; i < height_; i++) {
        for (int j = 0; j < width_; j++) {
            grid_[i][j] = false;
        }
    }

    // Place the pattern in the center
    for (int i = 0; i < patternHeight; i++) {
        for (int j = 0; j < patternWidth; j++) {
            int row = startRow + i;
            int col = startCol + j;
            if (row >= 0 && row < height_ && col >= 0 && col < width_) {
                grid_[row][col] = pattern[i][j];
            }
        }
    }
}

void GameOfLife::run() {
    while (running_) {
        // Clear screen
        clear();

        // Draw the current state
        draw();

        // Display generation count
        mvprintw(height_ + 1, 0, "Generation: %d", generation_);
        mvprintw(height_ + 2, 0, "Press ’q’ to quit, ’s’ to save image, ’r’ to randomize");

        // Process input
        int ch = getch();
        switch (ch) {
            case ’q’:
            case ’Q’:
                running_ = false;
                break;
            case ’s’:
            case ’S’:
                {
                    std::string filename = "gameoflife_" + generateTimestamp() + ".bmp";
                    if (saveAsBMP(filename)) {
                        mvprintw(height_ + 3, 0, "Image saved as %s", filename.c_str());
                        refresh();
                        std::this_thread::sleep_for(std::chrono::milliseconds(1000));
                    } else {
                        mvprintw(height_ + 3, 0, "Failed to save image");
                        refresh();
                        std::this_thread::sleep_for(std::chrono::milliseconds(1000));
                    }
                }
                break;
            case ’r’:
            case ’R’:
                initializeRandom();
                generation_ = 0;
                break;
        }

        // Update the game state
        update();
        generation_++;

        // Small delay to control game speed
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }
}

void GameOfLife::update() {
    // Calculate the next generation
    for (int i = 0; i < height_; i++) {
        for (int j = 0; j < width_; j++) {
            int neighbors = countNeighbors(i, j);

            // Apply Conway’s Game of Life rules
            if (grid_[i][j]) {
                // Cell is alive
                if (neighbors < 2 || neighbors > 3) {
                    nextGrid_[i][j] = false; // Cell dies
                } else {
                    nextGrid_[i][j] = true;  // Cell survives
                }
            } else {
                // Cell is dead
                if (neighbors == 3) {
                    nextGrid_[i][j] = true;  // Cell becomes alive
                } else {
                    nextGrid_[i][j] = false; // Cell stays dead
                }
            }
        }
    }

    // Update the grid with the new generation
    grid_ = nextGrid_;
}

int GameOfLife::countNeighbors(int row, int col) {
    int count = 0;

    // Check all 8 neighboring cells
    for (int i = -1; i <= 1; i++) {
        for (int j = -1; j <= 1; j++) {
            // Skip the cell itself
            if (i == 0 && j == 0) continue;

            // Calculate neighbor coordinates with wrapping
            int neighborRow = (row + i + height_) % height_;
            int neighborCol = (col + j + width_) % width_;

            // Count live neighbors
            if (grid_[neighborRow][neighborCol]) {
                count++;
            }
        }
    }

    return count;
}

void GameOfLife::draw() {
    for (int i = 0; i < height_; i++) {
        for (int j = 0; j < width_; j++) {
            if (grid_[i][j]) {
                // Cell is alive
                if (has_colors()) {
                    attron(COLOR_PAIR(1));
                    mvaddch(i, j, ’ ’);
                    attroff(COLOR_PAIR(1));
                } else {
                    mvaddch(i, j, ’#’);
                }
            } else {
                // Cell is dead
                if (has_colors()) {
                    attron(COLOR_PAIR(2));
                    mvaddch(i, j, ’ ’);
                    attroff(COLOR_PAIR(2));
                } else {
                    mvaddch(i, j, ’.’);
                }
            }
        }
    }

    // Update the screen
    refresh();
}

std::string GameOfLife::generateTimestamp() {
    auto now = std::time(nullptr);
    auto tm = *std::localtime(&now);

    std::ostringstream oss;
    oss « std::put_time(&tm, "%Y%m%d_%H%M%S");
    return oss.str();
}

bool GameOfLife::saveAsBMP(const std::string& filename) {
    std::ofstream file(filename, std::ios::binary);
    if (!file) {
        return false;
    }

    // Write BMP header
    writeBMPHeader(file, width_, height_);

    // Write BMP data
    writeBMPData(file);

    file.close();
    return true;
}

void GameOfLife::writeBMPHeader(std::ofstream& file, int width, int height) {
    // Calculate row size and padding
    int rowSize = ((width * 24 + 31) / 32) * 4;
    int paddingSize = rowSize - width * 3;
    int fileSize = 54 + rowSize * height;

    // BMP file header (14 bytes)
    unsigned char bmpFileHeader[14] = {
        'B', 'M',                       // Signature
        0, 0, 0, 0,                     // File size in bytes
        0, 0, 0, 0,                     // Reserved
        54, 0, 0, 0                     // Offset to start of pixel data
    };

    // Update file size in header
    bmpFileHeader[2] = (unsigned char)(fileSize);
    bmpFileHeader[3] = (unsigned char)(fileSize » 8);
    bmpFileHeader[4] = (unsigned char)(fileSize » 16);
    bmpFileHeader[5] = (unsigned char)(fileSize » 24);

    // BMP info header (40 bytes)
    unsigned char bmpInfoHeader[40] = {
        40, 0, 0, 0,                    // Info header size
        0, 0, 0, 0,                     // Width
        0, 0, 0, 0,                     // Height
        1, 0,                           // Number of color planes
        24, 0,                          // Bits per pixel
        0, 0, 0, 0,                     // Compression
        0, 0, 0, 0,                     // Image size
        0, 0, 0, 0,                     // X pixels per meter
        0, 0, 0, 0,                     // Y pixels per meter
        0, 0, 0, 0,                     // Colors in color table
        0, 0, 0, 0                      // Important color count
    };

    // Update width in header
    bmpInfoHeader[4] = (unsigned char)(width);
    bmpInfoHeader[5] = (unsigned char)(width » 8);
    bmpInfoHeader[6] = (unsigned char)(width » 16);
    bmpInfoHeader[7] = (unsigned char)(width » 24);

    // Update height in header
    bmpInfoHeader[8] = (unsigned char)(height);
    bmpInfoHeader[9] = (unsigned char)(height » 8);
    bmpInfoHeader[10] = (unsigned char)(height » 16);
    bmpInfoHeader[11] = (unsigned char)(height » 24);

    // Write headers
    file.write(reinterpret_cast<char*>(bmpFileHeader), 14);
    file.write(reinterpret_cast<char*>(bmpInfoHeader), 40);
}

void GameOfLife::writeBMPData(std::ofstream& file) {
    // Calculate row size and padding
    int rowSize = ((width_ * 24 + 31) / 32) * 4;
    int paddingSize = rowSize - width_ * 3;
    unsigned char padding[3] = {0, 0, 0};

    // BMP stores images bottom-up
    for (int i = height_ - 1; i >= 0; i--) {
        for (int j = 0; j < width_; j++) {
            unsigned char color[3];

            // Set pixel color (BGR format)
            if (grid_[i][j]) {
                // Live cell - green
                color[0] = 0;    // Blue
                color[1] = 255;  // Green
                color[2] = 0;    // Red
            } else {
                // Dead cell - black
                color[0] = 0;    // Blue
                color[1] = 0;    // Green
                color[2] = 0;    // Red
            }

            // Write pixel data
            file.write(reinterpret_cast<char*>(color), 3);
        }

        // Write padding
        if (paddingSize > 0) {
            file.write(reinterpret_cast<char*>(padding), paddingSize);
        }
    }
}
```

2.3 GameOfLife.h File Reference

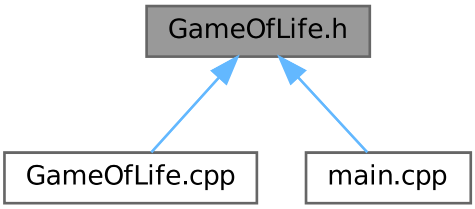
Conway's Game of Life implementation using ncurses.

```
#include <ncurses.h>
#include <vector>
#include <string>
#include <fstream>
#include <ctime>
#include <sstream>
#include <iomanip>
```

Include dependency graph for GameOfLife.h:



This graph shows which files directly or indirectly include this file:



Classes

- class GameOfLife

  Class implementing Conway's Game of Life simulation.

2.3.1 Detailed Description

Conway's Game of Life implementation using ncurses.

Author

Your Name

Date

March 2025

Definition in file GameOfLife.h.

2.4 GameOfLife.h

Go to the documentation of this file.

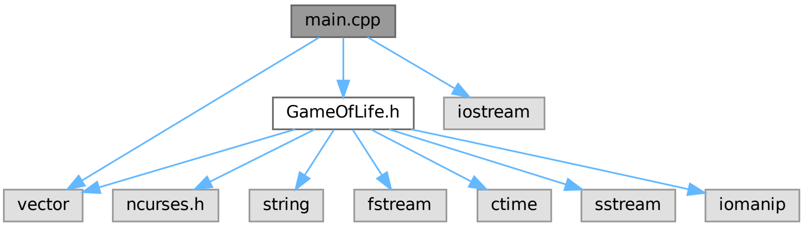
```
#ifndef GAME_OF_LIFE_H
#define GAME_OF_LIFE_H

#include <ncurses.h>
#include <vector>
#include <string>
#include <fstream>
#include <ctime>
#include <sstream>
#include <iomanip>

class GameOfLife {
public:
    GameOfLife(int height, int width);

    ~GameOfLife();

    void initializeRandom();

    void initializePattern(const std::vector<std::vector<bool»& pattern);

    void run();

    void update();

    void draw();

    bool saveAsBMP(const std::string& filename);

private:
    void initNCurses();

    int countNeighbors(int row, int col);

    std::string generateTimestamp();

    void writeBMPHeader(std::ofstream& file, int width, int height);

    void writeBMPData(std::ofstream& file);

    int height_;
    int width_;
    std::vector<std::vector<bool» grid_;
    std::vector<std::vector<bool» nextGrid_;
    bool running_;
    int generation_;
};

#endif // GAME_OF_LIFE_H

```

2.5 main.cpp File Reference

Main entry point for Conway's Game of Life.

```
#include "GameOfLife.h"
#include <iostream>
#include <vector>
```

Include dependency graph for main.cpp:



Functions

- std::vector< std::vector< bool > > createGlider ()

  Create a glider pattern.
- std::vector< std::vector< bool > > createBlinker ()

  Create a blinker pattern.
- std::vector< std::vector< bool > > createGosperGliderGun ()

  Create a Gosper glider gun pattern.
- int displayMenu ()

  Display menu and get user choice.
- int main ()

  Main function.

2.5.1 Detailed Description

Main entry point for Conway's Game of Life.

Author

Your Name

Date

March 2025

Definition in file main.cpp.

2.5.2 Function Documentation

createBlinker()

```
std::vector< std::vector< bool > > createBlinker ( )
```

Create a blinker pattern.

Returns

A vector representing a blinker pattern

Definition at line 29 of file main.cpp.

```
00029                                              {
00030       std::vector<std::vector<bool» pattern = {
00031           {false, false, false},
00032           {true, true, true},
00033           {false, false, false}
00034       };
00035       return pattern;
00036   }
```

createGlider()

```
std::vector< std::vector< bool > > createGlider ( )
```

Create a glider pattern.

Returns

A vector representing a glider pattern

Definition at line 16 of file main.cpp.

```
00016                                             {
00017       std::vector<std::vector<bool» pattern = {
00018           {false, true, false},
00019           {false, false, true},
00020           {true, true, true}
00021       };
00022       return pattern;
00023   }
```

createGosperGliderGun()

```
std::vector< std::vector< bool > > createGosperGliderGun ( )
```

Create a Gosper glider gun pattern.

Returns

A vector representing a Gosper glider gun pattern

Definition at line 42 of file main.cpp.

```
00042                                                {
00043     std::vector<std::vector<bool» pattern(11, std::vector<bool>(38, false));
00044 
00045     // Left block
00046     pattern[5][1] = true;
00047     pattern[6][1] = true;
00048     pattern[5][2] = true;
00049     pattern[6][2] = true;
00050 
00051     // Right block
00052     pattern[3][35] = true;
00053     pattern[4][35] = true;
00054     pattern[3][36] = true;
00055     pattern[4][36] = true;
00056 
00057     // Left ship
00058     pattern[3][13] = true;
00059     pattern[3][14] = true;
00060     pattern[4][12] = true;
00061     pattern[4][16] = true;
00062     pattern[5][11] = true;
00063     pattern[5][17] = true;
00064     pattern[6][11] = true;
00065     pattern[6][15] = true;
00066     pattern[6][17] = true;
00067     pattern[6][18] = true;
00068     pattern[7][11] = true;
00069     pattern[7][17] = true;
00070     pattern[8][12] = true;
00071     pattern[8][16] = true;
00072     pattern[9][13] = true;
00073     pattern[9][14] = true;
00074 
00075     // Right ship
00076     pattern[1][25] = true;
00077     pattern[2][23] = true;
00078     pattern[2][25] = true;
00079     pattern[3][21] = true;
00080     pattern[3][22] = true;
00081     pattern[4][21] = true;
00082     pattern[4][22] = true;
00083     pattern[5][21] = true;
00084     pattern[5][22] = true;
00085     pattern[6][23] = true;
00086     pattern[6][25] = true;
00087     pattern[7][25] = true;
00088 
00089     return pattern;
00090 }
```

displayMenu()

`int displayMenu ( )`

Display menu and get user choice.

Returns

User's menu choice

Definition at line 96 of file main.cpp.

```
00096                   {
00097     std::cout « "Conway’s Game of Life\n";
00098     std::cout « "=====================\n";
00099     std::cout « "1. Random pattern\n";
00100     std::cout « "2. Glider pattern\n";
00101     std::cout « "3. Blinker pattern\n";
00102     std::cout « "4. Gosper glider gun pattern\n";
00103     std::cout « "0. Exit\n";
00104     std::cout « "Enter your choice: ";
00105 
00106     int choice;
00107     std::cin » choice;
00108     return choice;
00109 }
```

main()

`int main ( )`

Main function.

Returns

Exit status

Definition at line 115 of file main.cpp.

```
00115              {
00116      int choice = displayMenu();
00117
00118      if (choice == 0) {
00119          return 0;
00120      }
00121
00122      // Create game with appropriate size
00123      int height = 30;
00124      int width = 80;
00125      GameOfLife game(height, width);
00126
00127      // Initialize based on user choice
00128      switch (choice) {
00129          case 1:
00130              game.initializeRandom();
00131              break;
00132          case 2:
00133              game.initializePattern(createGlider());
00134              break;
00135          case 3:
00136              game.initializePattern(createBlinker());
00137              break;
00138          case 4:
00139              game.initializePattern(createGosperGliderGun());
00140              break;
00141          default:
00142              game.initializeRandom();
00143              break;
00144      }
00145
00146      // Run the game
00147      game.run();
00148
00149      return 0;
00150  }
```

2.6 main.cpp

Go to the documentation of this file.

```

#include "GameOfLife.h"
#include <iostream>
#include <vector>

std::vector<std::vector<bool» createGlider() {
    std::vector<std::vector<bool» pattern = {
        {false, true, false},
        {false, false, true},
        {true, true, true}
    };
    return pattern;
}

std::vector<std::vector<bool» createBlinker() {
    std::vector<std::vector<bool» pattern = {
        {false, false, false},
        {true, true, true},
        {false, false, false}
    };
    return pattern;
}

std::vector<std::vector<bool» createGosperGliderGun() {
    std::vector<std::vector<bool» pattern(11, std::vector<bool>(38, false));

    // Left block
    pattern[5][1] = true;
    pattern[6][1] = true;
    pattern[5][2] = true;
    pattern[6][2] = true;

    // Right block
    pattern[3][35] = true;
    pattern[4][35] = true;
    pattern[3][36] = true;
    pattern[4][36] = true;

    // Left ship
    pattern[3][13] = true;
    pattern[3][14] = true;
    pattern[4][12] = true;
    pattern[4][16] = true;
    pattern[5][11] = true;
    pattern[5][17] = true;
    pattern[6][11] = true;
    pattern[6][15] = true;
    pattern[6][17] = true;
    pattern[6][18] = true;
    pattern[7][11] = true;
    pattern[7][17] = true;
    pattern[8][12] = true;
    pattern[8][16] = true;
    pattern[9][13] = true;
    pattern[9][14] = true;

    // Right ship
    pattern[1][25] = true;
    pattern[2][23] = true;
    pattern[2][25] = true;
    pattern[3][21] = true;
    pattern[3][22] = true;
    pattern[4][21] = true;
    pattern[4][22] = true;
    pattern[5][21] = true;
    pattern[5][22] = true;
    pattern[6][23] = true;
    pattern[6][25] = true;
    pattern[7][25] = true;

    return pattern;
}

int displayMenu() {
    std::cout « "Conway's Game of Life\n";
    std::cout « "=====================\n";
    std::cout « "1. Random pattern\n";
    std::cout « "2. Glider pattern\n";
    std::cout « "3. Blinker pattern\n";
    std::cout « "4. Gosper glider gun pattern\n";
    std::cout « "0. Exit\n";
    std::cout « "Enter your choice: ";

    int choice;
    std::cin » choice;
    return choice;
}

int main() {
    int choice = displayMenu();

    if (choice == 0) {
        return 0;
    }

    // Create game with appropriate size
    int height = 30;
    int width = 80;
    GameOfLife game(height, width);

    // Initialize based on user choice
    switch (choice) {
        case 1:
            game.initializeRandom();
            break;
        case 2:
            game.initializePattern(createGlider());
            break;
        case 3:
            game.initializePattern(createBlinker());
            break;
        case 4:
            game.initializePattern(createGosperGliderGun());
            break;
        default:
            game.initializeRandom();
            break;
    }

    // Run the game
    game.run();

    return 0;
}

```

Index