HW 8
Game of Life

Philip Nelson

2018 October 26

Introduction

The purpose of this assignment is to simulate Conway's Game of Life taking advantage of parallelization through MPI. The program begins by populating the world such that there is a one in five chance of a cell being alive. Then it splits the world up groups of rows and sends them to the other processes. Each process then exchanges information with the processes to the "north" (rank -) and to the "south" (rank + 1). They each determine the state of their strip for the next generation and send the results to the master to be displayed. This is repeated until the maximum number of generations is reached. The world can be displayed in ascii on the terminal, or saved as a png and combined into an animated gif image. The command line arguments are the world width, world height, number of generations to simulate and output type (0 - ascii, 1 - png/gif).

Code

The code is broken up into seven files, main.cpp, communication.hpp, rules.hpp, cell.hpp, random.hpp, output.hpp, and writePNG.hpp. The files are included below.

0.1 main.cpp

```
1  #include "cell.hpp"
2  #include "communication.hpp"
3  #include "output.hpp"
4  #include "random.hpp"
5  #include "rules.hpp"
6  #include <algorithm>
7  #include <iomanip>
8  #include <iostream>
9  #include <mpi.h>
10 #include <unistd.h>
11 #include <vector>
12
```

```
13 inline void help(std::string msg, int rank)
14 {
15   if (rank == 0)
16   {
17     std::cout << msg << std::endl;
18
19     std::cout
20       << "usage gameOfLife width height generations makeOutput
           outputType"
21       << std::endl;
22   }
23 }
24
25 inline void end()
26 {
27   MPI_Finalize();
28   exit(EXIT_SUCCESS);
29 }
30
31 int main(int argc, char** argv)
32 {
33   MPI_Init(&argc, &argv);
34
35   int rank, world_size;
36   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
37   MPI_Comm_size(MPI_COMM_WORLD, &world_size);
38
39   if (argc < 4)
40   {
41     help("", rank);
42     end();
43   }
44
45   auto shouldOutput = (bool)std::stoi(argv[4]);
46   Output output_type = Output::ASCII;
47   if (shouldOutput)
48   {
49     if (argc > 5)
50     {
51       output_type = static_cast<Output>(std::stoi(argv[5]));
52     }
53     else
54     {
55       help("Not enough arguments provided", rank);
56       end();
57     }
```

```
58    }
59
60    if (output_type > 1)
61    {
62      help("invalid outputType:\n  0 - term ascii\n  1 - gif\n", rank);
63      end();
64    }
65
66    auto width = std::stoi(argv[1]);
67    auto hight = std::stoi(argv[2]);
68    auto iters = std::stoi(argv[3]);
69    auto rpp = hight / world_size;
70    auto aspectRatio = (double)width / hight;
71    width = rpp * world_size;
72    hight = width / aspectRatio;
73
74    //
         ============================================================================//
75    //                                    Master
                                         //
76    //
         ============================================================================//
77    if (0 == rank)
78    {
79      std::cout << "Simulating:\n"
80                << "--------------------\n"
81                << "\n"
82                << width << " x " << hight << " world\n"
83                << "Processes: " << world_size << "\n"
84                << "Rows Per Process: " << rpp << "\n"
85                << "Make output: " << std::boolalpha << shouldOutput << "
                      \n"
86                << "--------------------\n"
87                << "\n";
88
89      /* setup world */
90      World world(hight);
91      std::for_each(begin(world), end(world), [&](std::vector<Cell>& row)
            {
92        for (auto i = 0; i < width; ++i)
93        {
94          row.push_back(random_int(0, 6) == 0 ? Cell::ALIVE : Cell::DEAD)
              ;
95        }
```

```
96      });
97
98      /* print the first generation */
99      if (shouldOutput) print_world(world, output_type);
100
101     /* send strips to other processes */
102     for (int dest = 1, row = rpp; dest < world_size; ++dest)
103     {
104       for (int tag = 0; tag < rpp; ++tag, ++row)
105       {
106         MPI_Send(world[row].data(), width, MPI_INT, dest, tag,
              MPI_COMM_WORLD);
107       }
108     }
109
110     /* copy out master's strip */
111     World strip(rpp + 2);
112     std::for_each(begin(strip), end(strip), [&](std::vector<Cell>& row)
          {
113       row.resize(width);
114     });
115     for (int row = 0; row < rpp; ++row)
116     {
117       strip[row + 1] = world[row];
118     }
119
120     /* Run the simulation */
121     auto simulationTime = 0.0;
122     auto imageTime = 0.0;
123     double t1, t2, t3;
124     //std::cout << "0\n";
125     for (auto i = 0; i < iters; ++i)
126     {
127       t1 = MPI_Wtime();
128     //std::cout << "1\n";
129
130       send_recv(strip, rank, world_size);
131       natural_selection(strip);
132       gatherMaster(world, strip, rpp, world_size);
133
134       t2 = MPI_Wtime();
135
136       if (shouldOutput)
137       {
138         print_world(world, output_type);
139         std::cout << "generation " << i << " complete\n";
```

```
140       }
141
142       t3 = MPI_Wtime();
143
144       simulationTime += t2 - t1;
145       imageTime += t3 - t2;
146       MPI_Barrier(MPI_COMM_WORLD);
147     }
148
149     /* convert the images in ./images/ to a gif using imagemagick */
150     auto t4 = MPI_Wtime();
151     if (shouldOutput)
152       system("convert -loop 0 -delay 25 `ls images | sort -g | sed "
153              "'s-^-images/-'` out.gif");
154     auto t5 = MPI_Wtime();
155     std::cout << "Simulation Time: " << simulationTime
156               << "\nImage Write Time: " << imageTime
157               << "\ngif Creating time: " << t5 - t4 << std::endl;
158   }
159   //
        ============================================================================//

160   //                                    Slave
                                          //
161   //
        ============================================================================//

162   else
163   {
164     World strip(rpp + 2);
165
166     /* resize rows to receive from neighbors */
167     std::for_each(begin(strip), end(strip), [&](std::vector<Cell>& row)
          {
168       row.resize(width);
169     });
170
171     /* receive strip from master */
172     MPI_Status stat;
173     for (auto row = 0; row < rpp; ++row)
174     {
175       MPI_Probe(0, MPI_ANY_TAG, MPI_COMM_WORLD, &stat);
176       MPI_Recv(strip[stat.MPI_TAG + 1].data(),
177                width,
178                MPI_INT,
179                0,
```

```
180                 MPI_ANY_TAG,
181                 MPI_COMM_WORLD,
182                 MPI_STATUS_IGNORE);
183     }
184
185     /* run the simulation */
186     for (auto i = 0; i < iters; ++i)
187     {
188       send_recv(strip, rank, world_size);
189       natural_selection(strip);
190       gatherSlave(strip, rpp);
191
192       MPI_Barrier(MPI_COMM_WORLD);
193     }
194   }
195
196   MPI_Finalize();
197
198   return EXIT_SUCCESS;
199 }
```

0.2 communication.hpp

```
1  #ifndef COMMUNICATION_HPP
2  #define COMMUNICATION_HPP
3
4  #include "cell.hpp"
5  #include <mpi.h>
6  #include <vector>
7
8  using World = std::vector<std::vector<Cell>>;
9
10 /**
11  * Share border information with neighbors
12  *
13  * @param world      The representation of the world known to the
       process
14  * @param rank       The process rank
15  * @param world_size The number of processes
16  */
17 void send_recv(World& world, int rank, int world_size)
18 {
19     //std::cout << rank << " -> 2\n";
20   auto destN = (rank - 1 < 0) ? world_size - 1 : rank - 1;
21     //std::cout << rank << " -> 3\n";
22   auto destS = (rank + 1) % world_size;
23     //std::cout << rank << " -> 4\n";
```

```
24    // std::cout << rank << " ^" << destN << " v" << destS << std::endl;
25    auto ct = world[1].size();
26      //std::cout << rank << " -> 5 " << world[1].size()<< " " << ct<< "
          " << MPI_INT<< " " << destN<< "\n";
27    /* clang-format off */
28    MPI_Request request1, request2;
29    MPI_Isend(
30      world[1].data(), ct, MPI_INT, destN, 0, MPI_COMM_WORLD, &request1);
31      //std::cout << rank << " -> 6\n";
32    MPI_Recv(
33      world[world.size()-1].data(), ct, MPI_INT, destS, 0, MPI_COMM_WORLD
          , MPI_STATUS_IGNORE);
34      //std::cout << rank << " -> 7\n";
35    MPI_Isend(
36      world[world.size()-2].data(), ct, MPI_INT, destS, 0, MPI_COMM_WORLD
          , &request2);
37      //std::cout << rank << " -> 8\n";
38    MPI_Recv(
39      world[0].data(), ct, MPI_INT, destN, 0, MPI_COMM_WORLD,
          MPI_STATUS_IGNORE);
40      //std::cout << rank << " -> 9\n";
41    /* clang-format on */
42    int flag1, flag2;
43    MPI_Test(&request1, &flag1, MPI_STATUS_IGNORE);
44    if(!flag1) std::cout << "REE1\n";
45    MPI_Test(&request1, &flag2, MPI_STATUS_IGNORE);
46    if(!flag2) std::cout << "REE2\n";
47  }
48
49  /**
50   * Gather all the strips to the master, (called by the master)
51   *
52   * @param world      The representation of the whole world
53   * @param strip      The representation of the world known to the
       process
54   * @param rpp        The number of rows given to each process
55   * @param world_size The number of processes
56   */
57  void gatherMaster(World& world, World& strip, int rpp, int world_size)
58  {
59    MPI_Status stat;
60    for (auto src = 1, row = rpp; src < world_size; ++src)
61    {
62      for (auto recvd = 0; recvd < rpp; ++recvd, ++row)
63      {
64        MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &stat);
```

```
65        auto r = stat.MPI_SOURCE * rpp + stat.MPI_TAG;
66        MPI_Recv(world[r].data(),
67                 world[r].size(),
68                 MPI_INT,
69                 stat.MPI_SOURCE,
70                 stat.MPI_TAG,
71                 MPI_COMM_WORLD,
72                 MPI_STATUS_IGNORE);
73      }
74    }
75    for (auto row = 0; row < rpp; ++row)
76    {
77      world[row] = strip[row + 1];
78    }
79  }
80
81  /**
82   * Gather all the strips to the master (called by the slaves)
83   *
84   * @param strip      The representation of the world known to the
       process
85   * @param rpp        The number of rows given to each process
86   */
87  void gatherSlave(World& strip, int rpp)
88  {
89    for (auto row = 0; row < rpp; ++row)
90    {
91      MPI_Send(strip[row + 1].data(),
92               strip[row + 1].size(),
93               MPI_INT,
94               0,
95               row,
96               MPI_COMM_WORLD);
97    }
98  }
99
100 #endif
```

0.3 rules.hpp

```
1  #ifndef RULES_HPP
2  #define RULES_HPP
3
4  #include "cell.hpp"
5  #include <vector>
6
7  /**
```

```
 8   * Count the number of neighbors for a cell in the world
 9   *
10   * @param world The representation of the whole world
11   * @param i     The row of the cell to check
12   * @param j     The column of the cell to check
13   * @return      The number of alive neighbors
14   */
15  int get_neighbors(std::vector<std::vector<Cell>> const& world,
16                    unsigned int i,
17                    unsigned int j)
18  {
19    int neighbors = 0;
20    if (j != world[i].size() - 1 && world[i][j + 1] == Cell::ALIVE)
21    {
22      ++neighbors;
23    }
24    if (j != 0 && world[i][j - 1] == Cell::ALIVE)
25    {
26      ++neighbors;
27    }
28    if (i != world.size() - 1 && j != world[i].size() - 1 &&
29        world[i + 1][j + 1] == Cell::ALIVE)
30    {
31      ++neighbors;
32    }
33    if (i != world.size() - 1 && world[i + 1][j] == Cell::ALIVE)
34    {
35      ++neighbors;
36    }
37    if (i != world.size() - 1 && j != 0 && world[i + 1][j - 1] == Cell::
          ALIVE)
38    {
39      ++neighbors;
40    }
41    if (i != 0 && j != world[i].size() - 1 && world[i - 1][j + 1] == Cell
          ::ALIVE)
42    {
43      ++neighbors;
44    }
45    if (i != 0 && world[i - 1][j] == Cell::ALIVE)
46    {
47      ++neighbors;
48    }
49    if (i != 0 && j != 0 && world[i - 1][j - 1] == Cell::ALIVE)
50    {
51      ++neighbors;
```

```
52    }
53    return neighbors;
54  }
55
56  /**
57   * determines if a cell is alive or dead based on it's current state
        and the
58   * number of neighbors
59   *
60   * @param neighbors The number of living neighbors
61   * @param state     The current state of a cell
62   * @return          The state of the cell in the next generation
63   */
64  Cell live_die(int neighbors, Cell state)
65  {
66    if (state == Cell::ALIVE)
67    {
68      if (neighbors < 2)
69      {
70        return Cell::DEAD;
71      }
72      else if (neighbors > 3)
73      {
74        return Cell::DEAD;
75      }
76      else
77      {
78        return Cell::ALIVE;
79      }
80    }
81    if (state == Cell::DEAD)
82    {
83      if (neighbors == 3)
84      {
85        return Cell::ALIVE;
86      }
87      else
88      {
89        return Cell::DEAD;
90      }
91    }
92    return Cell::DEAD;
93  }
94
95  /**
96   * determines the state of every cell in the world for the next
```

```
     generation
 *
 * @param world The representation of the whole world
 */
void natural_selection(std::vector<std::vector<Cell>>& world)
{
  std::vector<std::vector<Cell>> next_gen = world;
  for (auto i = 0u; i < world.size(); ++i)
    for (auto j = 0u; j < world[i].size(); ++j)
      next_gen[i][j] = (live_die(get_neighbors(world, i, j), world[i][j
          ]));
  world = next_gen;
}
#endif
```

0.4 cell.hpp

```
#ifndef CELL_HPP
#define CELL_HPP

/**
 * The enumeration of cell states
 */
enum Cell
{
  ALIVE, /**< The alive state */
  DEAD   /**< The dead state */
};

#endif
```

0.5 random.hpp

```
#ifndef RANDOM_HPP
#define RANDOM_HPP

#include <algorithm>
#include <functional>
#include <random>

/**
 * Generate a random number from [low, high]
 *
 * @param low  The lower bound
 * @param high The upper bound
 * @return     A random number on the range [low, high]
 */
```

```
15 int random_int(int low, int high)
16 {
17   static std::random_device rd;
18   static std::mt19937 mt(rd());
19   std::uniform_int_distribution<> dist(low, high);
20   return dist(mt);
21 }
22
23 /**
24  * Generate a random number from [low, high)
25  *
26  * @param low  The lower bound
27  * @param high The upper bound
28  * @return     A random number on the range [low, high)
29  */
30 double random_double(double low, double high)
31 {
32   static std::random_device rd;
33   static std::mt19937 mt(rd());
34   std::uniform_real_distribution<> dist(low, high);
35   return dist(mt);
36 }
37
38 /**
39  * Fill a container from [first, last) with random numbers from [low,
       high]
40  *
41  * @param first Iterator to beginning of range to fill
42  * @param last  Iterator to end of range to fill
43  * @param low   The lower bound
44  * @param high  The upper bound
45  */
46 template <typename it>
47 void random_int_fill(it first, it last, const int low, const int high)
48 {
49   static std::random_device rd;
50   static std::mt19937 mt(rd());
51   std::uniform_int_distribution<> dist(low, high);
52   std::generate(first, last, std::bind(dist, mt));
53 }
54
55 /**
56  * Fill a container from [first, last) with random numbers from [low,
       high)
57  *
58  * @param first Iterator to beginning of range to fill
```

```
59   * @param last  Iterator to end of range to fill
60   * @param low   The lower bound
61   * @param high  The upper bound
62   */
63  template <typename it>
64  void random_double_fill(it first, it last, const double low, const
        double high)
65  {
66    static std::random_device rd;
67    static std::mt19937 mt(rd());
68    std::uniform_real_distribution<double> dist(low, high);
69    std::generate(first, last, std::bind(dist, mt));
70  }
71
72  #endif
```

0.6 output.hpp

```
1   #ifndef OUTPUT_HPP
2   #define OUTPUT_HPP
3
4   #include "cell.hpp"
5   #include "writePNG.hpp"
6   #include <algorithm>
7   #include <iomanip>
8   #include <iostream>
9   #include <unistd.h>
10  #include <vector>
11
12  enum Output
13  {
14    ASCII, /**< print in ascii to the terminal */
15    GIF    /**< save as sequentially named pngs and converted to a gif */
16  };
17
18  using World = std::vector<std::vector<Cell>>;
19
20  /**
21   * Print a row of the world in ascii to the terminal
22   *
23   * @param row The row of the world to print
24   */
25  void print_ascii_row(std::vector<Cell> const& row)
26  {
27    for (auto i = 0u; i < row.size(); ++i)
28    {
29      std::cout << (row[i] == Cell::ALIVE ? "*" : ".");
```

```
30    }
31    std::cout << '\n';
32  }
33
34  /**
35   * Print the world in ascii to the terminal
36   *
37   * @param world The representation of the whole world
38   */
39  void print_ascii_world(World const& world)
40  {
41    std::cout << "\033[2J\033[1;1H";
42    for (auto i = 0u; i < world.size(); ++i)
43    {
44      for (auto j = 0u; j < world[0].size(); ++j)
45      {
46        std::cout << (world[i][j] == Cell::ALIVE ? "*" : ".");
47      }
48      std::cout << '\n';
49    }
50    std::cout << std::setw(world[0].size()) << std::setfill('-') << '-'
51              << std::endl;
52  }
53
54  /**
55   * Save the world as a png image
56   *
57   * @param world The representation of the whole world
58   */
59  void print_png_world(World const& world)
60  {
61    static int image_num = 0;
62    int scale = 3;
63    std::vector<uint8_t> image;
64    std::for_each(begin(world), end(world), [&](auto row) {
65      for (int l = 0; l < scale; ++l)
66      {
67        std::for_each(begin(row), end(row), [&](auto c) {
68          auto color = c == Cell::ALIVE ? 0 : 255;
69          for (auto i = 0; i < scale; ++i)
70          {
71            image.push_back(color);
72            image.push_back(color);
73            image.push_back(color);
74          }
75        });
```

```
76      }
77    });
78    std::string filename = "images/" + std::to_string(++image_num) + ".
          png";
79    save_png_libpng(
80      filename, image.data(), world[0].size() * scale, world.size() *
            scale);
81  }
82
83  /**
84   * Print the world in the given output type
85   *
86   * @param world       The representation of the whole world
87   * @param output_type The output type
88   */
89  void print_world(World world, Output output_type)
90  {
91    switch (output_type)
92    {
93    case Output::ASCII:
94      print_ascii_world(world);
95      sleep(1);
96      break;
97    case Output::GIF:
98      print_png_world(world);
99      break;
100   }
101 }
102
103 #endif
```

0.7 writePNG.hpp

```
1   #ifndef WRITE_PNG_HPP
2   #define WRITE_PNG_HPP
3
4   #include <png.h>
5   #include <string>
6
7   /**
8    * Saves a pixel buffer as a png using libpng
9    *
10   * @param filename The name to save the image to
11   * @param pixels   The buffer of pixel data
12   * @param w        The width of the image
13   * @param h        The height of the image
14   */
```

```
15 bool save_png_libpng(const std::string filename, uint8_t* pixels, int w
       , int h)
16 {
17   png_structp png =
18     png_create_write_struct(PNG_LIBPNG_VER_STRING, nullptr, nullptr,
         nullptr);
19   if (!png)
20   {
21     return false;
22   }
23
24   png_infop info = png_create_info_struct(png);
25   if (!info)
26   {
27     png_destroy_write_struct(&png, &info);
28     return false;
29   }
30
31   FILE* fp = fopen(filename.c_str(), "wb");
32   if (!fp)
33   {
34     png_destroy_write_struct(&png, &info);
35     return false;
36   }
37
38   png_init_io(png, fp);
39   png_set_IHDR(png,
40                info,
41                w,
42                h,
43                8 /* depth */,
44                PNG_COLOR_TYPE_RGB,
45                PNG_INTERLACE_NONE,
46                PNG_COMPRESSION_TYPE_BASE,
47                PNG_FILTER_TYPE_BASE);
48   png_colorp palette =
49     (png_colorp)png_malloc(png, PNG_MAX_PALETTE_LENGTH * sizeof(
         png_color));
50   if (!palette)
51   {
52     fclose(fp);
53     png_destroy_write_struct(&png, &info);
54     return false;
55   }
56   png_set_PLTE(png, info, palette, PNG_MAX_PALETTE_LENGTH);
57   png_write_info(png, info);
```

```
58    png_set_packing(png);
59
60    png_bytepp rows = (png_bytepp)png_malloc(png, h * sizeof(png_bytep));
61    for (int i = 0; i < h; ++i)
62    {
63      rows[i] = (png_bytep)(pixels + (h - i) * w * 3);
64    }
65
66    png_write_image(png, rows);
67    png_write_end(png, info);
68    png_free(png, palette);
69    png_destroy_write_struct(&png, &info);
70
71    fclose(fp);
72    delete[] rows;
73    return true;
74  }
75
76  #endif
```

Output

```
# mpic++ -O3 -lpng main.cpp -o gameOfLife.out

# mpiexec --oversubscribe -n 5 release 1024 1024 25 1
```

Findings

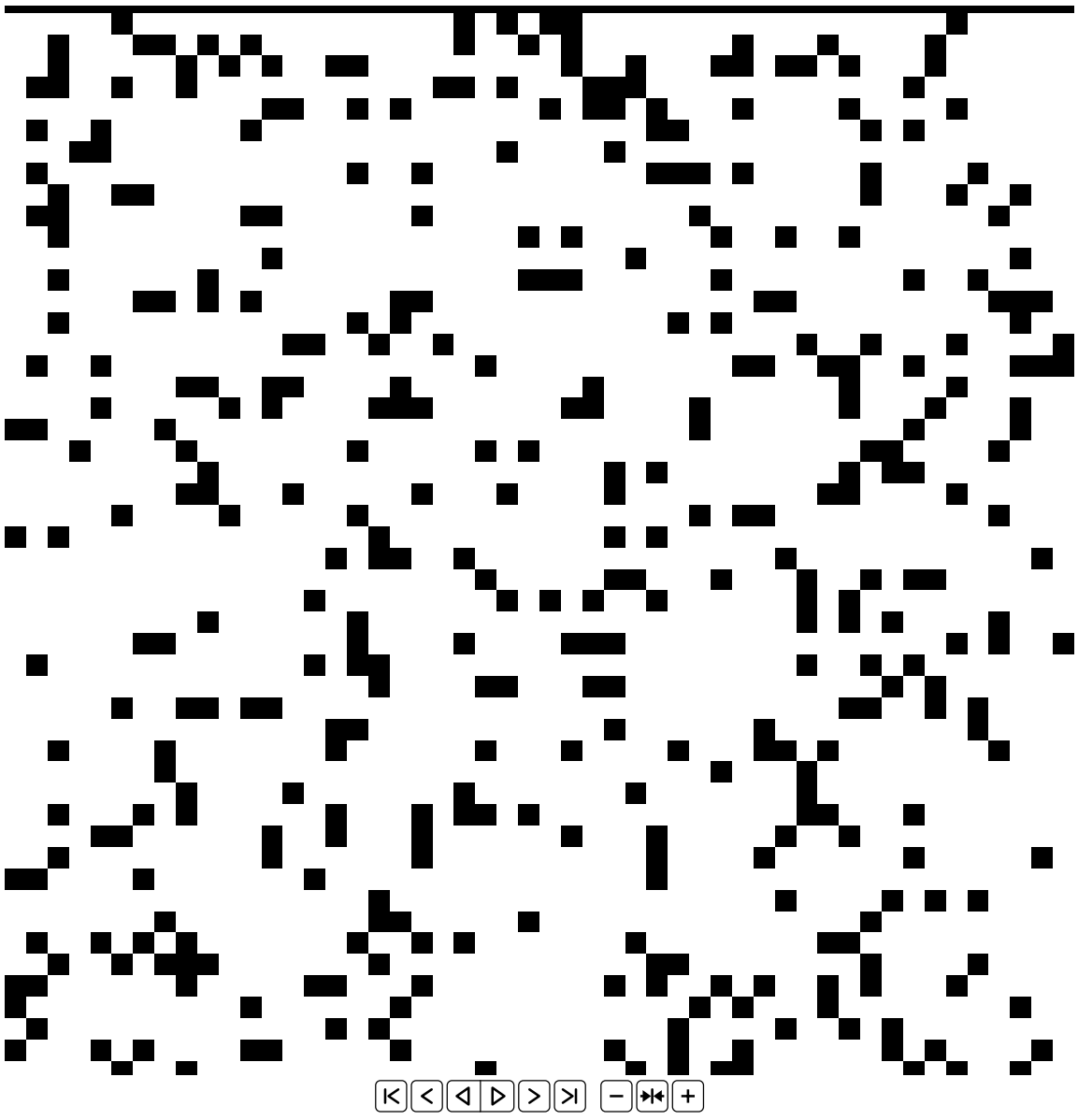

Figure 1: Conway's Game of Life