**TECHNOLOGICAL UNIVERSITY DUBLIN**

**Simulating the Sharks and Fishes Problem**

by

Dahir Mussa , Michael Dillon, Thomas Fokas and Shannon

Dunne

A project report submitted for

Parallel Computing



in the

School of Computing

Department of Informatics and Engineering

December 2020

# Declaration of Authorship

“I am aware of the University policy on plagiarism in assignments and examinations (3AS08). I understand that plagiarism, collusion, and copying are grave and serious offences in the University and I will accept the penalties that could be imposed if I engage in any such activity. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I declare that this material, which I now submit for assessment, is entirely of my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work.”

TECHNOLOGICAL UNIVERSITY DUBLIN

# *Abstract*

School of Computing

Department of Informatics and Engineering

Bachelor of Science in Computer Information

by Dahir Mussa, Michael Dillon, Thomas Fokas and Shannon Dunne

This document is a report on Simulating the ’Sharks and Fishes’ problem in a Parallel Computing Environment. The ’Sharks and Fishes’ problem is a classic ’predator-prey’ model of cellular automata. It will describe all the stages of the project. The elements discussed within this report will be:

Contents

[Declaration of Authorship 2](#_Toc59131294)

[*Abstract* 3](#_Toc59131295)

[1.1 Background 5](#_Toc59131296)

[1.1.1 Classical Model 5](#_Toc59131297)

[1.1.2 Cellular Automata 6](#_Toc59131298)

[1.2 Our work 6](#_Toc59131299)

[1.3 Overview of the Report 6](#_Toc59131300)

[2.1 Serial 7](#_Toc59131301)

[2.2 Parallel 7](#_Toc59131302)

[4.1 Rules of The Simulation 10](#_Toc59131303)

[4.2 Results of your simulations 12](#_Toc59131304)

[4.2.1 Generation One 12](#_Toc59131305)

[4.2.2 Generation One Thousand 13](#_Toc59131306)

[4.2.3 Generation Five Thousand 14](#_Toc59131307)

[4.2.4 Generation Ten Thousand 15](#_Toc59131308)

[4.2.5 Generation Twenty Thousand 16](#_Toc59131309)

[5.1 Conclusions 17](#_Toc59131310)

[5.2 Future Work 17](#_Toc59131311)

[7.1 Personal Reflections 19](#_Toc59131312)

[7.1.1 Dahir Mussa 19](#_Toc59131313)

[7.1.2 Michael Dillon 19](#_Toc59131314)

[7.1.3 Thomas Fokas 19](#_Toc59131315)

[7.1.4 Shannon Dunne 19](#_Toc59131316)

[7.2 Serial Programming Code 20](#_Toc59131317)

[7.1 sharks \_ fish.c 20](#_Toc59131318)

[7.2 newOcean.c 21](#_Toc59131319)

[7.3 checkStates.c 22](#_Toc59131320)

[7.4 checkNeighbours.c 30](#_Toc59131321)

[7.5 header.h 32](#_Toc59131322)

[7.6 Makefile 32](#_Toc59131323)

[7.7 Parallel Programming Code 33](#_Toc59131324)

[7.8 sharks \_ fish.c 33](#_Toc59131325)

[7.9 newOcean.c 35](#_Toc59131326)

[7.10 checkStates.c 36](#_Toc59131327)

[7.11 checkNeighbours.c 37](#_Toc59131328)

[7.12 header.h 37](#_Toc59131329)

[7.13 Makefile 37](#_Toc59131330)

[7.14 Parallel.c 37](#_Toc59131331)

[7.15 Visualisation Programming Code 37](#_Toc59131332)

[7.16 sharks \_ fish.c 37](#_Toc59131333)

[7.17 newOcean.c 37](#_Toc59131334)

[7.18 checkStates.c 37](#_Toc59131335)

[7.19 checkNeighbours.c 37](#_Toc59131336)

**Chapter 1**

**Introduction**

The objective of this project is to the “Sharks and Fishes” problem is a classic “predator prey” problem that uses the concept of cellular automata (see below). In this problem, the Sharks (predators) hunt and eat Fishes (prey) in a two-dimensional sea. Both Sharks and Fishes move, eat, give birth and die in the sea according to a set of predefined rules. The aim of this project is to write a program to simulate and visualise this using the concepts of cellular automata.

The project was to deliver 3 realizations: a serial execution program of our model, one running in parallel and a visualization program to display the results of the simulation.

# 1.1 Background

The ‘Sharks and Fishes’ problem; is an example of a Cellular Automata, utilizing the ‘Predator-Prey Model; which is based on the work of Bill Madden, Nancy Ricca, and Jonathan Rizzo, whilst at Montclair State University. This is achieved by placing each cell in a state. Each cell is then affected by their neighbouring cells, according to the rules that have been set by the developer, meaning either a cell can either: breed a fish/shark, that fish/shark could die, or the fish/shark already living within the cell could age up. Each cell is affected simultaneously in a generation corresponding with the rules; set by the developer. These rules are then repeated over several generations, changing the states of cells each generation.

# 1.1.1 Classical Model

The purpose of the classical model is a massive shift of labour demand or labour supply that is caused by a change in work preferences, technology, or labour market incentives.

“The major assumption of this model is that the economy is always at full employment” [4], which will have everyone working and using resources to the fullest of capabilities. The logic behind it is that if competition can work, the economy will be towards full employment. “Remember what happened when traffic slowed down because there was to many cars? After a few minutes, everything went back to normal.” [4], leads to believe the economy is adaptive to the flow of the flow of things and that it needs no help from anyone to correct.

# 1.1.2 Cellular Automata

The purpose of cellular automaton is discrete space and time, they describe the local interactions using rules for each selected point of cell. Cellular automaton is a mathematical model that is suited for a complex system, that contains a large number of components with local interactions.[5]. They were first created in 1940’s by John von Neumann.

A cellular automaton is one of the famous models, for example, Conway’s game of life. Where they are 2 states live or dead depending on the rules with 2-dimensional array, there are two types of neighbourhoods the von Neumann neighbourhood and Moore neighbourhood [6].

# 1.2 Our work

We worked as a team, and we had meetings near daily, each member of the team was assigned a task. All the members in the team had implemented some of the C Program for both: the serial and parallel programs, and also the visualization program, written in Java.

# 1.3 Overview of the Report

Our project is based on the ’sharks and fishes’ problem in a parallel computing environment. Firstly, we developed a serial program, we created a 2-dimensional array representing the sea. Using the C programming language, we implemented rules for the sharks and fish such as eat, give birth, and die.

The main goal was to parallelize and make it more efficient. Finally, we developed visualisation application using the java programming language which allowed the viewing of the sharks and fish problem, in order for us to develop the visualisation we used the serial code and change it into java.

**Chapter 2**

**Program Design**

# 2.1 Serial

Serial computing is when a computer program is broken up into several different instructions. These instructions are then processed and executed one at a time, by a single processor, ’singular fashion’ [3] until all the instructions have been executed.

Serial programming is much slower and less cost-effective than parallel programming. Therefore, serial programs cannot reasonably compete with parallel programs for solving large and complex problems [3]

# 2.2 Parallel

Parallel computing is when a program is broken up into separate sections each section is then broken up into several instructions. In parallel computing, multiple of the section’s instructions can be executed at the same time on multiple processors [3]. The primary goal of parallel computing is to increase the available computation power so that the application processing and problem-solving is faster [4].Parallel computing is when a program is broken up into separate sections each section is then broken up into several instructions. In parallel computing, multiple of the section’s instructions can be executed at the same time on multiple processors [3]. The primary goal of parallel computing is to increase the available computation power so that the application processing and problem-solving is faster [4].

**Chapter 3**

**Testing**

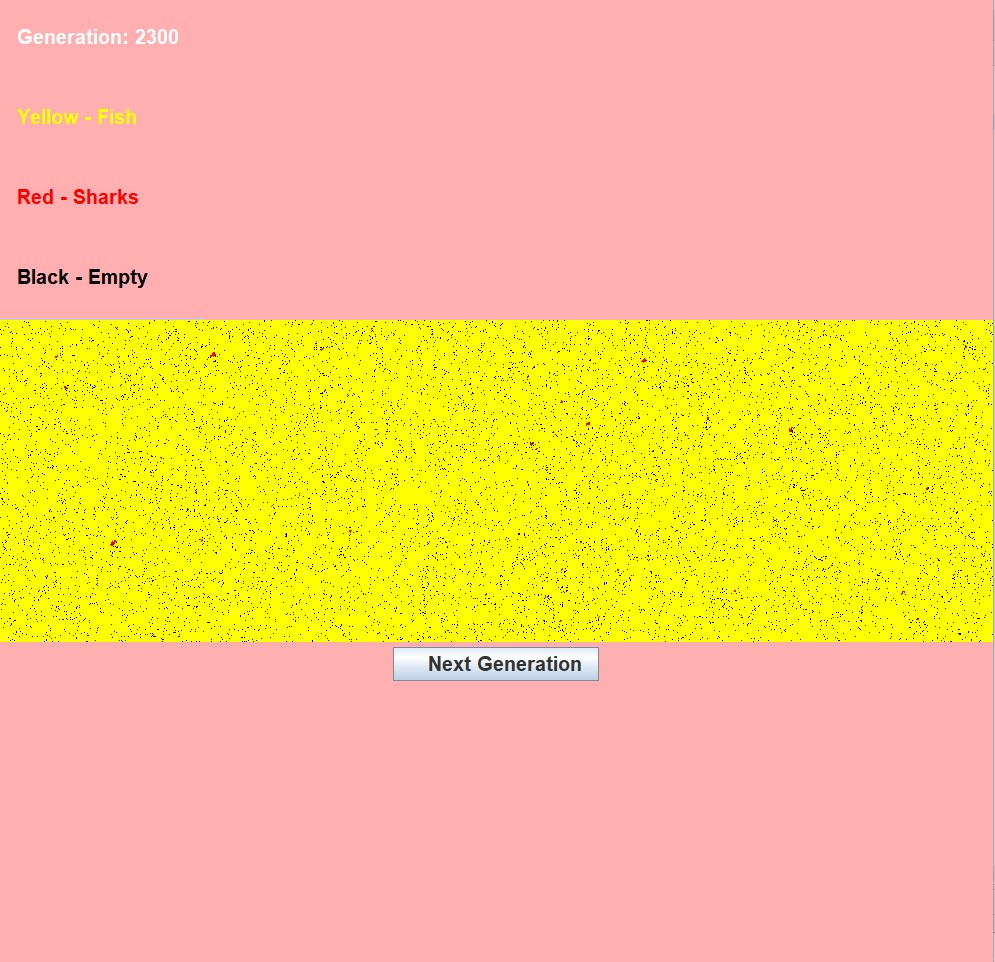
We tested the rules quite extensively. Thus, we came across rulesets that completely wiped out sharks, and/or fishes entirely. The most interesting rule by far was the ’5% chance of death per shark’ rule.



As you can see in the image above there are no/very few fish left.

Therefore, proving the need for the rule ’5% chance of death per shark’, as without this

rule the ocean/grid is not sustainable. Interestingly increasing this 5% to 10% has a huge impact of the shark population, as you can see below.



This mere 5% sends the shark population to dangerously low.

**Chapter 4**

**Results**

# 4.1 Rules of The Simulation

The Ocean Starting Rules:

* Approximately 45% of the ocean is ’Adult Fish’.
* Approximately 15% of the ocean is ’Baby Fish’.
* Approximately 15% of the ocean is ’Adult Sharks’.
* Approximately 15% of the ocean is ’Baby Sharks’.
* Approximately 20% of the ocean is ’Empty’.

Death Rules (Shark)

* Sharks live up to 20 generations each.
* Each shark has a 5% chance of death per generation.
* Each shark has a 50% chance of death if they have no fish neighbours (simulating starvation).
* Each shark has a 30% chance of death if they have 5 shark neighbours, this also increases by 15% per extra shark neighbour (up to 75%).

Fish Death Rules

* Fish live up to 10 generations each.
* Each fish has a 10% chance of death per shark neighbour (maximum number of shark neighbours is 8).
* Each fish has a 5% chance of death, if they have 5 fish neighbours. This increases by 5% per fish neighbour.
* Breeding Rules
* Shark nor Fish can breed if they are not at least 3 generations old.
* Must have a breed-able neighbour (of the same species) and a mutual empty neighbour cell.

# 4.2 Results of your simulations

Below are results of a 1000x1000 grid.

Yellow represents fish. Red represents sharks. Black represents empty cells.

# 4.2.1 Generation One



# 4.2.2 Generation One Thousand



# 4.2.3 Generation Five Thousand



# 4.2.4 Generation Ten Thousand



# 4.2.5 Generation Twenty Thousand



**Chapter 5**

**Conclusions and Future Work**

# 5.1 Conclusions

To conclude, we observed that the parallel program is faster than the serial program, this is because serial programming is a process in which one task is completed at a time, while parallel programming is a process in which multiple tasks are completed simultaneously.

The project is not fully completed, many improvements could have been made but we are quite happy with the provided results, in that they showed how efficient parallel programming van be.

# 5.2 Future Work

There are many improvements to be made within the project. We could not compare our serial program with our parallel program, as we did not figure out how to get it working in time, but if we had more time in the future, we would aim to improve this. We also did not get the parallel code to run correctly as we did not add the correct function to parallel the code. We tried different functions but could not get it to run. If we had more time, we believe we would have achieved this. we would also aim to improve the visualization program by making it read data files, to get the result and also adding a parallel part. Many improvements could be made, if we had more time, we would of made these improvements.

**Chapter 6**

**References**

1. Guru99.com. 2020. What Is C Programming Language? Basics, Introduction And History. [online] Available at: ¡https://www.guru99.com/c-programming-language.html¿ [Accessed 2 October 2020].
2. Techopedia.com. 2020.What Is C (Programming Language)? – Definition From Techopedia. [online] Available at: ¡https://www.techopedia.com/definition/24068/cprogramminglanguagec¿ [Accessed 2 October 2020].
3. Abhay B. Rathod, Rajratna Khadse, M Faruk Bagwan, initial. (2014) ‘SERIALCOMPUTING vs. PARALLEL COMPUTING: A COMPARATIVE STUDY USING MATLAB’ International Journal of Computer Science and Mobile Computing, Vol. 3, Issue 5, pg.815-820
4. Grama, A., Gupta, A., Karypis, G. and Kumar, V. (2003). Introduction to ParallelComputing. Harlow, England: Pearson Addison Wesley.
5. El Yacoubi, S. and El Jai, A., 2020. *Cellular Automata Modelling And Spreadability*. .[oline] Available at:<<https://www.sciencedirect.com/science/article/pii/S0895717702002595>>
6. Arxiv.org. 2020. [online] Available at: <https://arxiv.org/ftp/arxiv/papers/1811/1811.12387.pdf> >[Accessed 17 November 2020].

**Appendix**

# 7.1 Personal Reflections

# 7.1.1 Dahir Mussa

”I enjoyed this project; I learned a lot from the project and working with the team members and I also learned new skills and improved on some of the skills already possessed. I improved my knowledge on the c programming language.”

# 7.1.2 Michael Dillon

”I enjoyed working on this project, I feel I learned a lot throughout the project, such as: ’C-Programming’, ’The advantages of Parallel computing’, and perhaps the most useful skill ’Teamwork’.”

# 7.1.3 Thomas Fokas

”I enjoyed this project as I liked working in a team. I feel like my team I not only learned from the lectures I also learned from my team members.”

# 7.1.4 Shannon Dunne

”I enjoyed working on this project with my team, as I found the project subject matter quite interesting. I especially enjoyed the near daily team meetings. I also feel like I learned a lot more this year than last year

# 7.2 Serial Programming Code

# 7.1 sharks \_ fish.c

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include "header.h"  // Michael Dillon (B00120328), Dahir Mussa (B00107811), Shannon Dunne (B00095740), Thomas Fokas (B00121134)  // global variables.  int a;  int i;  int j;  int numberofgenerations = 20;  int Cells[N][N];  int adultFish = 0;  int adultShark = 0;  int babyFish = 0;  int babyShark = 0;  int main()  {    srand((unsigned int)time(NULL));    newOcean(); // Randomly Generated, a new ocean    for(a = 1; a <= numberofgenerations; a++)  {  fprintf(fptr, "Generation: %d\n", a);    for(i = 1; i < N - 1; i++) // [100][100]  {  for(j = 1; j < N - 1; j++)  {  checkNeighbours();    if((Cells[i][j] > 2) && (Cells[i][j] < 11))  {  adultFish += 1;  }  else if((Cells[i][j] > 12) && (Cells[i][j] < 30))  {  adultShark += 1;  }  else if((Cells[i][j] < 3) && (Cells[i][j] > 0))  {  babyFish += 1;  }  else if((Cells[i][j] < 13) && (Cells[i][j] > 10))  {  babyShark += 1;  }    print("%d ", Cells[i][j]);    checkCellStates();  }    printf("\n");  }    printf("\n");    printf("There are %d fish.\n", adultFish);  printf("There are %d sharks.\n", adultShark);  printf("There are %d baby fish.\n", babyFish);  printf("There are %d baby sharks.\n\n", babyShark);    adultFish = 0;  adultShark = 0;  babyFish = 0;  babyShark = 0;  }  } |

# 7.2 newOcean.c

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include "header.h"  extern int i;  extern int j;  extern int Cells[N][N];  // Roughly 20 % Sharks, 5 % being children (who can not yet breed)  // Roughly 60 % Fish, 15 % being children (who can not yet breed)  // Roughly 20 % empty  void newOcean()  {  for(i = 1; i < N - 1; i++)  {  for(j = 1; j < N - 1; j++)  {  float rand\_number = ( (float) rand() ) / RAND\_MAX;    if(rand\_number <= 0.15)  {  Cells[i][j] = 13; // Adult Shark  }  else if((rand\_number > 0.15) && (rand\_number <= 0.20))  {  Cells[i][j] = 11; // Baby Shark  }  else if((rand\_number > 0.20) && (rand\_number <= 0.65))  {  Cells[i][j] = 3; // Adult Fish  }  else if((rand\_number > 0.65) && (rand\_number <= 0.80))  {  Cells[i][j] = 1; // Baby Fish  }  else if(rand\_number > 0.80)  {  Cells[i][j] = 0; // Empty  }  }  }  } |

# 7.3 checkStates.c

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include "header.h"  extern int i;  extern int j;  extern int Cells[N][N];  extern int fishNeighbor;  extern int sharkNeighbor;  void checkCellStates()  {  // Generate Random Float Point Number  float rand\_number = ( (float) rand() ) / RAND\_MAX;  // Fish Code  if((Cells[i][j] > 0) && (Cells[i][j] < 11)) // Meaning Fish  {  if(Cells[i][j] > 2) // Fish Breeding  {  if((Cells[i - 1][j - 1] > 2) && (Cells[i - 1][j - 1] < 11)) // Row Above [i][j] // i-1 is North, j-1 is West... > 2 AND < 11 as fish can only breed between 3 and 10  {  if(Cells[i - 1][j] == 0) // 0 meaning empty  {  Cells[i - 1][j] = 1;  }  else if(Cells[i][j - 1] == 0) // if Cells[i - 1][j] is not empty try here!  {  Cells[i][j - 1] = 1;  }  }  else if((Cells[i - 1][j] > 2) && (Cells[i - 1][j] < 11)) // Row Above [i][j]  {  if(Cells[i - 1][j - 1] == 0)  {  Cells[i - 1][j - 1] = 1;  }  else if(Cells[i - 1][j + 1] == 0)  {  Cells[i - 1][j + 1] = 1;  }  else if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 1;  }  else if(Cells[i][j + 1] == 0)  {  Cells[i][j + 1] = 1;  }  }  else if((Cells[i - 1][j + 1] > 2) && (Cells[i - 1][j + 1] < 11)) // Row Above [i][j]  {  if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 1;  }  else if(Cells[i][j - 1] == 0)  {  Cells[i][j + 1] = 1;  }  }  else if((Cells[i][j - 1] > 2) && (Cells[i][j - 1] < 11)) // Row Of [i][j]  {  if(Cells[i - 1][j - 1] == 0)  {  Cells[i - 1][j - 1] = 1;  }  else if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 1;  }  else if(Cells[i + 1][j - 1] == 0)  {  Cells[i + 1][j - 1] = 1;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 1;  }  }  else if((Cells[i][j + 1] > 2) && (Cells[i][j + 1] < 11)) // Row Of [i][j]  {  if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 1;  }  else if(Cells[i - 1][j + 1] == 0)  {  Cells[i - 1][j + 1] = 1;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 1;  }  else if(Cells[i + 1][j + 1] == 0)  {  Cells[i + 1][j + 1] = 1;  }  }  else if((Cells[i + 1][j - 1] > 2) && (Cells[i + 1][j - 1] < 11)) // Row Below [i][j]  {  if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 1;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 1;  }  }  else if((Cells[i + 1][j] > 2) && (Cells[i + 1][j] < 11))  {  if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 1;  }  else if(Cells[i][j + 1] == 0)  {  Cells[i][j + 1] = 1;  }  else if(Cells[i + 1][j - 1] == 0)  {  Cells[i + 1][j - 1] = 1;  }  else if(Cells[i + 1][j + 1] == 0)  {  Cells[i + 1][j + 1] = 1;  }  }  else if((Cells[i + 1][j + 1] > 2) && (Cells[i + 1][j + 1] < 11))  {  if(Cells[i][j + 1] == 0)  {  Cells[i][j + 1] = 1;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 1;  }  }  } // End Of Fish Breeding Age!    // Shark Killing Fish    // if Fish has 1 Shark Neightbor, Fish has 12.5% chance of death.    if((sharkNeighbor == 1) && (rand\_number <= 0.125))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 2) && (rand\_number <= 0.25))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 3) && (rand\_number <= 0.375))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 4) && (rand\_number <= 0.50))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 5) && (rand\_number <= 0.625))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 6) && (rand\_number <= 0.75))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 7) && (rand\_number <= 0.825))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 8) && (rand\_number <= 1))  {  Cells[i][j] = 31;  }    // Fish Overcrowding Death    if((fishNeighbor == 4) && (rand\_number <= 0.15))  {  Cells[i][j] = 31;  }  else if((fishNeighbor == 5) && (rand\_number <= 0.3))  {  Cells[i][j] = 31;  }  else if((fishNeighbor == 6) && (rand\_number <= 0.45))  {  Cells[i][j] = 31;  }  else if((fishNeighbor == 7) && (rand\_number <= 0.6))  {  Cells[i][j] = 31;  }  else if((fishNeighbor == 8) && (rand\_number <= 0.75))  {  Cells[i][j] = 31;  }    // Fish Live To 10, Therefore while Cell[i][j] is not 10, the fish will age (persuming they haven been eaten!)    if(Cells[i][j] != 10) // If the fish is not 10, it will age up  {  Cells[i][j] += 1; // If the fish is one --- it will be two nect gen, If the fish is two --- it will be three next gen (and can now breed)  }  else // If the fish is 10, it will die  {  Cells[i][j] = 31;  }  }  // Shark Code    if((Cells[i][j] > 10) && (Cells[i][j] < 31)) // Meaning Shark  {  if(Cells[i][j] > 12) // Breeding Age  {  if((Cells[i - 1][j - 1] > 12) && (Cells[i - 1][j - 1] < 31)) // Row Above [i][j]  {  if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 11;  }  else if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 11;  }  }  else if((Cells[i - 1][j] > 12) && (Cells[i - 1][j] < 31)) // Row Above [i][j]  {  if(Cells[i - 1][j - 1] == 0)  {  Cells[i - 1][j - 1] = 11;  }  else if(Cells[i - 1][j + 1] == 0)  {  Cells[i - 1][j + 1] = 11;  }  else if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 11;  }  else if(Cells[i][j + 1] == 0)  {  Cells[i][j + 1] = 11;  }  }  else if((Cells[i - 1][j + 1] > 12) && (Cells[i - 1][j + 1] < 31)) // Row Above [i][j]  {  if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 11;  }  else if(Cells[i][j - 1] == 0)  {  Cells[i][j + 1] = 11;  }  }  else if((Cells[i][j - 1] > 12) && (Cells[i][j - 1] < 31)) // Row Of [i][j]  {  if(Cells[i - 1][j - 1] == 0)  {  Cells[i - 1][j - 1] = 11;  }  else if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 11;  }  else if(Cells[i + 1][j - 1] == 0)  {  Cells[i + 1][j - 1] = 11;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 11;  }  }  else if((Cells[i][j + 1] > 12) && (Cells[i][j + 1] < 31)) // Row Of [i][j]  {  if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 11;  }  else if(Cells[i - 1][j + 1] == 0)  {  Cells[i - 1][j + 1] = 11;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 11;  }  else if(Cells[i + 1][j + 1] == 0)  {  Cells[i + 1][j + 1] = 11;  }  }  else if((Cells[i + 1][j - 1] > 12) && (Cells[i + 1][j - 1] < 31)) // Row Below [i][j]  {  if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 11;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 11;  }  }  else if((Cells[i + 1][j] > 12) && (Cells[i + 1][j] < 31))  {  if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 11;  }  else if(Cells[i][j + 1] == 0)  {  Cells[i][j + 1] = 11;  }  else if(Cells[i + 1][j - 1] == 0)  {  Cells[i + 1][j - 1] = 11;  }  else if(Cells[i + 1][j + 1] == 0)  {  Cells[i + 1][j + 1] = 11;  }  }  else if((Cells[i + 1][j + 1] > 12) && (Cells[i + 1][j + 1] < 31))  {  if(Cells[i][j + 1] == 0)  {  Cells[i][j + 1] = 11;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 11;  }  }  } // End of Shark Breeding Age!    if((Cells[i][j] > 10 && Cells[i][j] < 31) && (rand\_number <= 0.03))  {  Cells[i][j] = 32; // 32 = Dead Shark  }    if((fishNeighbor == 0) && (rand\_number <= 0.30))  {  Cells[i][j] = 32; // 32 = Dead Shark, Shark died due to no food.  }  if(Cells[i][j] < 30)  {  Cells[i][j] += 1;  }  else  {  Cells[i][j] = 32;  }  }  if(Cells[i][j] == 31) // Dead Fish  {  Cells[i][j] = 0; // Empty The Cell  }  if(Cells[i][j] == 32) // Dead Shark  {  Cells[i][j] = 0; // Empty Cell  }  } |

# 7.4 checkNeighbours.c

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include "header.h"  extern int Cells[N][N];  extern int i;  extern int j;  int sharkNeighbor = 0;  int fishNeighbor = 0;  void checkNeighbours()  {  // North West  if((Cells[i-1][j-1] > 0) && (Cells[i-1][j-1] < 11))  {  fishNeighbor += 1;  }  else if((Cells[i-1][j-1] > 10) && (Cells[i-1][j-1] < 31))  {  sharkNeighbor += 1;  }    // North  if((Cells[i-1][j] > 0) && ((Cells[i-1][j] < 11)))  {  fishNeighbor += 1;  }  else if((Cells[i-1][j] > 10) && ((Cells[i-1][j] < 31)))  {  sharkNeighbor += 1;  }    // North East  if((Cells[i-1][j+1] > 0) && ((Cells[i-1][j] < 11)))  {  fishNeighbor += 1;  }  else if((Cells[i-1][j+1] > 10) && ((Cells[i-1][j] < 31)))  {  sharkNeighbor += 1;  }    // West  if((Cells[i][j-1] > 0) && ((Cells[i-1][j] < 11)))  {  fishNeighbor += 1;  }  else if((Cells[i][j-1] > 10) && ((Cells[i-1][j] < 31)))  {  sharkNeighbor += 1;  }    // East  if((Cells[i][j+1] > 0) && (Cells[i-1][j] < 11))  {  fishNeighbor += 1;  }  else if((Cells[i][j+1] > 10) && (Cells[i-1][j] < 31))  {  sharkNeighbor += 1;  }    // South West  if((Cells[i+1][j-1] > 0) && (Cells[i-1][j] < 11))  {  fishNeighbor += 1;  }  else if((Cells[i+1][j-1] > 10) && (Cells[i-1][j] < 31))  {  sharkNeighbor += 1;  }    // South  if((Cells[i+1][j] > 0) && (Cells[i-1][j] < 11))  {  fishNeighbor += 1;  }  else if((Cells[i+1][j] > 10) && (Cells[i-1][j] < 31))  {  sharkNeighbor += 1;  }  // South East  if((Cells[i+1][j+1] > 0) && (Cells[i-1][j] < 11))  {  fishNeighbor += 1;  }  else if((Cells[i+1][j+1] > 10) && (Cells[i-1][j] < 31))  {  sharkNeighbor += 1;  }  } |

# 7.5 header.h

|  |
| --- |
| #define N 20  void checkNeighbours();  void checkCellStates();  void newOcean(); |

# 7.6 Makefile

|  |
| --- |
| sharks\_fish: sharks\_fish.o checkStates.o checkNeighbours.o newOcean.o  gcc sharks\_fish.o checkStates.o checkNeighbours.o newOcean.o -o sharks\_fish  sharks\_fish.o: sharks\_fish.c  gcc -c sharks\_fish.c  checkCellStates.o: checkStates.c  gcc -c checkStates.c  checkNeighbours.o: checkNeighbours.c  gcc -c checkNeighbours.c  newOcean.o: newOcean.c  gcc -c newOcean.c  clean:  rm \*.o sharks\_fish |

# 

# 7.7 Parallel Programming Code

# 7.8 sharks \_ fish.c

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>  #include <time.h>  #include "timer.h"  #include "header.h" //#include "libmainprogram.h"//  // Michael Dillon (B00120328), Dahir Mussa (B00107811), Shannon Dunne (B00095740), Thomas Fokas (B00121134)  // global variables.  int a;  int i;  int j;  int numberofgenerations = 20;  int Cells[N][N];  int adultFish = 0;  int adultShark = 0;  int babyFish = 0;  int babyShark = 0;  int thread\_count = 4;  int counter;  double elapsed\_time;  double my\_start , my\_finish , my\_elapsed;  pthread\_mutex\_t barrier\_mutex;  void\* parallel(void\* rank);  int main(int argc,char \*argv[])  {  pthread\_mutex\_init(&barrier\_mutex,NULL);    long thread;  pthread\_t\* thread\_handles;  if(argc == 2)  {  thread\_count = atoll(argv[1]);  }  else  {  printf("sorry you didnt enter any numbers of threads , please enter any numbers of thread that you wnat\n");  return -1;  }  thread\_handles = malloc(thread\_count \* sizeof(pthread\_t));    srand((unsigned int)time(NULL));    newOcean(); // Randomly Generated, a new ocean    for(a = 1; a <= numberofgenerations; a++)  {  printf("Generation: %d\n", a); // %d means print integer, in this case a. \n new line    for(i = 1; i < N - 1; i++) // [100][100]  {  for(j = 1; j < N - 1; j++)  {  checkNeighbours();    if((Cells[i][j] > 2) && (Cells[i][j] < 11))  {  adultFish += 1;  }  else if((Cells[i][j] > 12) && (Cells[i][j] < 30))  {  adultShark += 1;  }  else if((Cells[i][j] < 3) && (Cells[i][j] > 0))  {  babyFish += 1;  }  else if((Cells[i][j] < 13) && (Cells[i][j] > 10))  {  babyShark += 1;  }    printf("%d\t", Cells[i][j]);    checkCellStates();      GET\_TIME(my\_start);    for(thread = 0; thread< thread\_count;thread++)  pthread\_create(&thread\_handles[thread], NULL, parallel, (void \*) thread);    for(thread = 0; thread < thread\_count; thread++)  pthread\_join(thread\_handles[thread], NULL);    GET\_TIME(my\_finish);  my\_elapsed = my\_finish - my\_start;    }  printf("\n");  }    printf("\n");    printf("There are %d fish.\n", adultFish);  printf("There are %d sharks.\n", adultShark);  printf("There are %d baby fish.\n", babyFish);  printf("There are %d baby sharks.\n\n", babyShark);    adultFish = 0;  adultShark = 0;  babyFish = 0;  babyShark = 0;  }    printf("\nTime it took for the thread to complete = %e seconds\n\n" ,my\_elapsed);    pthread\_mutex\_destroy(&barrier\_mutex);  free(thread\_handles);    return 0;  } |

# 7.9 newOcean.c

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include "header.h"  #include "timer.h"  extern int i;  extern int j;  extern int Cells[N][N];  // Roughly 20 % Sharks, 5 % being children (who can not yet breed)  // Roughly 60 % Fish, 15 % being children (who can not yet breed)  // Roughly 20 % empty  void newOcean()  {  for(i = 1; i < N - 1; i++)  {  for(j = 1; j < N - 1; j++)  {  float rand\_number = ( (float) rand() ) / RAND\_MAX;    if(rand\_number <= 0.15)  {  Cells[i][j] = 13; // Adult Shark  }  else if((rand\_number > 0.15) && (rand\_number <= 0.20))  {  Cells[i][j] = 11; // Baby Shark  }  else if((rand\_number > 0.20) && (rand\_number <= 0.65))  {  Cells[i][j] = 3; // Adult Fish  }  else if((rand\_number > 0.65) && (rand\_number <= 0.80))  {  Cells[i][j] = 1; // Baby Fish  }  else if(rand\_number > 0.80)  {  Cells[i][j] = 0; // Empty  }  }  }  } |

# 7.10 checkStates.c

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include "header.h"  #include "timer.h"  extern int i;  extern int j;  extern int Cells[N][N];  extern int fishNeighbor;  extern int sharkNeighbor;  void checkCellStates()  {  // Generate Random Float Point Number  float rand\_number = ( (float) rand() ) / RAND\_MAX;  // Fish Code  if((Cells[i][j] > 0) && (Cells[i][j] < 11)) // Meaning Fish  {  if(Cells[i][j] > 2) // Fish Breeding  {  if((Cells[i - 1][j - 1] > 2) && (Cells[i - 1][j - 1] < 11)) // Row Above [i][j] // i-1 is North, j-1 is West... > 2 AND < 11 as fish can only breed between 3 and 10  {  if(Cells[i - 1][j] == 0) // 0 meaning empty  {  Cells[i - 1][j] = 1;  }  else if(Cells[i][j - 1] == 0) // if Cells[i - 1][j] is not empty try here!  {  Cells[i][j - 1] = 1;  }  }  else if((Cells[i - 1][j] > 2) && (Cells[i - 1][j] < 11)) // Row Above [i][j]  {  if(Cells[i - 1][j - 1] == 0)  {  Cells[i - 1][j - 1] = 1;  }  else if(Cells[i - 1][j + 1] == 0)  {  Cells[i - 1][j + 1] = 1;  }  else if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 1;  }  else if(Cells[i][j + 1] == 0)  {  Cells[i][j + 1] = 1;  }  }  else if((Cells[i - 1][j + 1] > 2) && (Cells[i - 1][j + 1] < 11)) // Row Above [i][j]  {  if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 1;  }  else if(Cells[i][j - 1] == 0)  {  Cells[i][j + 1] = 1;  }  }  else if((Cells[i][j - 1] > 2) && (Cells[i][j - 1] < 11)) // Row Of [i][j]  {  if(Cells[i - 1][j - 1] == 0)  {  Cells[i - 1][j - 1] = 1;  }  else if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 1;  }  else if(Cells[i + 1][j - 1] == 0)  {  Cells[i + 1][j - 1] = 1;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 1;  }  }  else if((Cells[i][j + 1] > 2) && (Cells[i][j + 1] < 11)) // Row Of [i][j]  {  if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 1;  }  else if(Cells[i - 1][j + 1] == 0)  {  Cells[i - 1][j + 1] = 1;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 1;  }  else if(Cells[i + 1][j + 1] == 0)  {  Cells[i + 1][j + 1] = 1;  }  }  else if((Cells[i + 1][j - 1] > 2) && (Cells[i + 1][j - 1] < 11)) // Row Below [i][j]  {  if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 1;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 1;  }  }  else if((Cells[i + 1][j] > 2) && (Cells[i + 1][j] < 11))  {  if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 1;  }  else if(Cells[i][j + 1] == 0)  {  Cells[i][j + 1] = 1;  }  else if(Cells[i + 1][j - 1] == 0)  {  Cells[i + 1][j - 1] = 1;  }  else if(Cells[i + 1][j + 1] == 0)  {  Cells[i + 1][j + 1] = 1;  }  }  else if((Cells[i + 1][j + 1] > 2) && (Cells[i + 1][j + 1] < 11))  {  if(Cells[i][j + 1] == 0)  {  Cells[i][j + 1] = 1;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 1;  }  }  } // End Of Fish Breeding Age!    // Shark Killing Fish    // if Fish has 1 Shark Neightbor, Fish has 12.5% chance of death.    if((sharkNeighbor == 1) && (rand\_number <= 0.125))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 2) && (rand\_number <= 0.25))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 3) && (rand\_number <= 0.375))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 4) && (rand\_number <= 0.50))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 5) && (rand\_number <= 0.625))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 6) && (rand\_number <= 0.75))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 7) && (rand\_number <= 0.825))  {  Cells[i][j] = 31;  }  else if((sharkNeighbor == 8) && (rand\_number <= 1))  {  Cells[i][j] = 31;  }    // Fish Overcrowding Death    if((fishNeighbor == 4) && (rand\_number <= 0.15))  {  Cells[i][j] = 31;  }  else if((fishNeighbor == 5) && (rand\_number <= 0.3))  {  Cells[i][j] = 31;  }  else if((fishNeighbor == 6) && (rand\_number <= 0.45))  {  Cells[i][j] = 31;  }  else if((fishNeighbor == 7) && (rand\_number <= 0.6))  {  Cells[i][j] = 31;  }  else if((fishNeighbor == 8) && (rand\_number <= 0.75))  {  Cells[i][j] = 31;  }  if(Cells[i][j] != 10) // If the fish is not 10, it will age up  {  Cells[i][j] += 1;  }  else // If the fish is 10, it will die  {  Cells[i][j] = 31;  }  }  // Shark Code    if((Cells[i][j] > 10) && (Cells[i][j] < 31)) // Meaning Shark  {  if(Cells[i][j] > 12) // Breeding Age  {  if((Cells[i - 1][j - 1] > 12) && (Cells[i - 1][j - 1] < 31)) // Row Above [i][j]  {  if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 11;  }  else if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 11;  }  }  else if((Cells[i - 1][j] > 12) && (Cells[i - 1][j] < 31)) // Row Above [i][j]  {  if(Cells[i - 1][j - 1] == 0)  {  Cells[i - 1][j - 1] = 11;  }  else if(Cells[i - 1][j + 1] == 0)  {  Cells[i - 1][j + 1] = 11;  }  else if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 11;  }  else if(Cells[i][j + 1] == 0)  {  Cells[i][j + 1] = 11;  }  }  else if((Cells[i - 1][j + 1] > 12) && (Cells[i - 1][j + 1] < 31)) // Row Above [i][j]  {  if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 11;  }  else if(Cells[i][j - 1] == 0)  {  Cells[i][j + 1] = 11;  }  }  else if((Cells[i][j - 1] > 12) && (Cells[i][j - 1] < 31)) // Row Of [i][j]  {  if(Cells[i - 1][j - 1] == 0)  {  Cells[i - 1][j - 1] = 11;  }  else if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 11;  }  else if(Cells[i + 1][j - 1] == 0)  {  Cells[i + 1][j - 1] = 11;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 11;  }  }  else if((Cells[i][j + 1] > 12) && (Cells[i][j + 1] < 31)) // Row Of [i][j]  {  if(Cells[i - 1][j] == 0)  {  Cells[i - 1][j] = 11;  }  else if(Cells[i - 1][j + 1] == 0)  {  Cells[i - 1][j + 1] = 11;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 11;  }  else if(Cells[i + 1][j + 1] == 0)  {  Cells[i + 1][j + 1] = 11;  }  }  else if((Cells[i + 1][j - 1] > 12) && (Cells[i + 1][j - 1] < 31))  {  if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 11;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 11;  }  }  else if((Cells[i + 1][j] > 12) && (Cells[i + 1][j] < 31))  {  if(Cells[i][j - 1] == 0)  {  Cells[i][j - 1] = 11;  }  else if(Cells[i][j + 1] == 0)  {  Cells[i][j + 1] = 11;  }  else if(Cells[i + 1][j - 1] == 0)  {  Cells[i + 1][j - 1] = 11;  }  else if(Cells[i + 1][j + 1] == 0)  {  Cells[i + 1][j + 1] = 11;  }  }  else if((Cells[i + 1][j + 1] > 12) && (Cells[i + 1][j + 1] < 31))  {  if(Cells[i][j + 1] == 0)  {  Cells[i][j + 1] = 11;  }  else if(Cells[i + 1][j] == 0)  {  Cells[i + 1][j] = 11;  }  }  } // End of Shark Breeding Age!    if((Cells[i][j] > 10 && Cells[i][j] < 31) && (rand\_number <= 0.03))  {  Cells[i][j] = 32; // 32 = Dead Shark  }    if((fishNeighbor == 0) && (rand\_number <= 0.30)) // 30 Percent chance to starve  {  Cells[i][j] = 32; // 32 = Dead Shark, Shark died due to no food.  }    if(Cells[i][j] < 30)  {  Cells[i][j] += 1;  }  else  {  Cells[i][j] = 32;  }  }  if(Cells[i][j] == 31) // Dead Fish  {  Cells[i][j] = 0; // Empty The Cell  }  if(Cells[i][j] == 32) // Dead Shark  {  Cells[i][j] = 0; // Empty Cell  }  } |

# 7.11 checkNeighbours.c

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include "header.h"  #include "timer.h"  extern int Cells[N][N];  extern int i;  extern int j;  int sharkNeighbor = 0;  int fishNeighbor = 0;  void checkNeighbours()  {  // North West  if((Cells[i-1][j-1] > 0) && (Cells[i-1][j-1] < 11))  {  fishNeighbor += 1;  }  else if((Cells[i-1][j-1] > 10) && (Cells[i-1][j-1] < 31))  {  sharkNeighbor += 1;  }    // North  if((Cells[i-1][j] > 0) && ((Cells[i-1][j] < 11)))  {  fishNeighbor += 1;  }  else if((Cells[i-1][j] > 10) && ((Cells[i-1][j] < 31)))  {  sharkNeighbor += 1;  }    // North East  if((Cells[i-1][j+1] > 0) && ((Cells[i-1][j] < 11)))  {  fishNeighbor += 1;  }  else if((Cells[i-1][j+1] > 10) && ((Cells[i-1][j] < 31)))  {  sharkNeighbor += 1;  }    // West  if((Cells[i][j-1] > 0) && ((Cells[i-1][j] < 11)))  {  fishNeighbor += 1;  }  else if((Cells[i][j-1] > 10) && ((Cells[i-1][j] < 31)))  {  sharkNeighbor += 1;  }    // East  if((Cells[i][j+1] > 0) && (Cells[i-1][j] < 11))  {  fishNeighbor += 1;  }  else if((Cells[i][j+1] > 10) && (Cells[i-1][j] < 31))  {  sharkNeighbor += 1;  }    // South West  if((Cells[i+1][j-1] > 0) && (Cells[i-1][j] < 11))  {  fishNeighbor += 1;  }  else if((Cells[i+1][j-1] > 10) && (Cells[i-1][j] < 31))  {  sharkNeighbor += 1;  }    // South  if((Cells[i+1][j] > 0) && (Cells[i-1][j] < 11))  {  fishNeighbor += 1;  }  else if((Cells[i+1][j] > 10) && (Cells[i-1][j] < 31))  {  sharkNeighbor += 1;  }  // South East  if((Cells[i+1][j+1] > 0) && (Cells[i-1][j] < 11))  {  fishNeighbor += 1;  }  else if((Cells[i+1][j+1] > 10) && (Cells[i-1][j] < 31))  {  sharkNeighbor += 1;  }  } |

# 7.12 header.h

|  |
| --- |
| #define N 22  void checkNeighbours();  void checkCellStates();  void newOcean(); |

# 7.13 Makefile

|  |
| --- |
| sharks\_fish: sharks\_fish.o checkStates.o checkNeighbours.o newOcean.o parallel.o header.h  gcc sharks\_fish.o checkStates.o checkNeighbours.o newOcean.o parallel.o -o sharks\_fish -lpthread  sharks\_fish.o: sharks\_fish.c  gcc -c sharks\_fish.c  checkCellStates.o: checkStates.c  gcc -c checkStates.c  checkNeighbours.o: checkNeighbours.c  gcc -c checkNeighbours.c  newOcean.o: newOcean.c  gcc -c newOcean.c  parallel.o: parallel.c  gcc -c parallel.c  clean:  rm \*.o sharks\_fish |

# 7.14 Parallel.c

|  |
| --- |
| #include <pthread.h>  #include <math.h>  #include "header.h"  #include "timer.h"  #include <stdio.h>  //#include "parallel.h"  extern long thread\_count;  extern int counter;  extern pthread\_mutex\_t barrier\_mutex;  void \*parallel(void\* rank)  {  long my\_rank = (long) rank;    int local\_m = N/thread\_count;  int my\_first\_row = my\_rank\*local\_m;  int my\_last\_row = (my\_rank + 1) \* local\_m - 1;  int i;    for(i = my\_first\_row; i < my\_last\_row; i++);    pthread\_mutex\_lock(&barrier\_mutex);  counter++;  pthread\_mutex\_unlock(&barrier\_mutex);  while(counter < thread\_count);  } |

# 

# 

# 7.15 Visualisation Programming Code

# 7.16 sharks \_ fish.c

|  |
| --- |
| import java.util.Random;  import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JLabel;  import javax.swing.JPanel;  import java.awt.Color;  import java.awt.Font;  import java.awt.Graphics;  import java.awt.GridLayout;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  // Michael Dillon (B00120328), Dahir Mussa(B00107811), Shannon Dunne(B00095740), Thomas Fokas(B00121134)  public class program extends JFrame implements ActionListener  {  // Array  static int n = 1000;  static int Cells[][] = new int[n][n];    static int i = 1;  static int j = 1;    // Generations  static int a = 1;    static float f;    static int adultFish;  static int adultShark;  static int babyFish;  static int babyShark;    // Random Float  static Random rand = new Random();    JPanel genPan = new JPanel();  JPanel butPan = new JPanel();    JButton button = new JButton(" Next Generation");    // Counters  JLabel jLabel = new JLabel(" Generation: " + a);    JLabel jLabel2 = new JLabel(" Yellow - Fish");  JLabel jLabel3 = new JLabel(" Red - Sharks");  JLabel jLabel4 = new JLabel(" Black - Empty");    public program()  {  new newOcean();    // Design  Font font = new Font("Arial", Font.BOLD, 20);    jLabel.setForeground(Color.WHITE);  jLabel.setFont(font);  jLabel2.setForeground(Color.yellow);  jLabel2.setFont(font);    jLabel3.setForeground(Color.red);  jLabel3.setFont(font);    jLabel4.setForeground(Color.black);  jLabel4.setFont(font);    genPan.setBackground(Color.pink);  butPan.setBackground(Color.pink);    genPan.setLayout(new GridLayout(4, 1));  setLayout(new GridLayout(3, 1));    button.setFont(font);    genPan.add(jLabel);  genPan.add(jLabel2);  genPan.add(jLabel3);  genPan.add(jLabel4);    // Adding Components    button.addActionListener(this);  butPan.add(button);    add(genPan);  add(visPan);  add(butPan);    setDefaultCloseOperation(EXIT\_ON\_CLOSE);  setResizable(false);  setSize(n, n);  setTitle("Sharks And Fishes");  setVisible(true);  }    JPanel visPan = new JPanel(){  public void paint(Graphics g)  {  for(i = 1; i < n - 1; i++)  {  for(j = 1; j < n - 1; j++)  {  program.f = program.rand.nextFloat();    new checkNeighbors();  new checkStates();    if((Cells[i][j] > 0) && (Cells[i][j] < 3))  {  g.setColor(Color.YELLOW);  }  else if((Cells[i][j] > 2) && (Cells[i][j] < 11))  {  g.setColor(Color.YELLOW);  }  else if((Cells[i][j] > 10) && (Cells[i][j] < 13))  {  g.setColor(Color.RED);  }    else if((Cells[i][j] > 12) && (Cells[i][j] < 31))  {  g.setColor(Color.RED);  }  else  {  g.setColor(Color.BLACK);  }  g.fillRect(i, j, 1, 1);  }  }  }  };    public static void main(String[] args)  {  new program();  }  @Override  public void actionPerformed(ActionEvent e)  {  a++;  jLabel.setText(" Generation: " + a);  visPan.repaint();  }  } |

# 

# 7.17 newOcean.c

|  |
| --- |
| public class newOcean  {  public newOcean()  {  for (int i = 1; i < program.n - 1; i++)  {  for (int j = 1; j < program.n - 1; j++)  {  float f = program.rand.nextFloat();    if(f < 0.45)  {  program.adultFish += 1;  program.Cells[i][j] = 3;  }  else if(f < 0.60)  {  program.babyFish += 1;  program.Cells[i][j] = 1;  }  else if(f < 0.75)  {  program.adultShark += 1;  program.Cells[i][j] = 13;  }  else if(f < 0.80)  {  program.babyShark += 1;  program.Cells[i][j] = 11;  }  else  {  program.Cells[i][j] = 0;  }  }  }  }  } |

# 

# 7.18 checkStates.c

|  |
| --- |
| public class checkStates  {  public checkStates()  {  // Meaning Fish  if((program.Cells[program.i][program.j] > 0) && (program.Cells[program.i][program.j] < 11))  {  // Breeding Rules  if(program.Cells[program.i][program.j] > 2)  {  if((program.Cells[program.i - 1][program.j - 1] > 2) && (program.Cells[program.i - 1][program.j - 1] < 11)) // NW... DONE  {  if(program.Cells[program.i - 1][program.j] == 0)  {  program.Cells[program.i - 1][program.j] = 1;  }  else if(program.Cells[program.i][program.j - 1] == 0)  {  program.Cells[program.i][program.j - 1] = 1;  }  }  else if((program.Cells[program.i - 1][program.j] > 2) && (program.Cells[program.i - 1][program.j] < 11)) // N... Done  {  if(program.Cells[program.i - 1][program.j - 1] == 0)  {  program.Cells[program.i - 1][program.j - 1] = 1;  }  else if(program.Cells[program.i - 1][program.j + 1] == 0)  {  program.Cells[program.i - 1][program.j + 1] = 1;  }  else if(program.Cells[program.i][program.j - 1] == 0)  {  program.Cells[program.i][program.j - 1] = 1;  }  else if(program.Cells[program.i][program.j + 1] == 0)  {  program.Cells[program.i][program.j + 1] = 1;  }  }  else if((program.Cells[program.i - 1][program.j + 1] > 2) && (program.Cells[program.i - 1][program.j + 1] < 11)) // NE... Done  {  if(program.Cells[program.i - 1][program.j] == 0)  {  program.Cells[program.i - 1][program.j] = 1;  }  else if(program.Cells[program.i][program.j + 1] == 0)  {  program.Cells[program.i][program.j + 1] = 1;  }  }  else if((program.Cells[program.i][program.j + 1] > 2) && (program.Cells[program.i][program.j + 1] < 11)) // E... Done  {  if(program.Cells[program.i - 1][program.j] == 0)  {  program.Cells[program.i - 1][program.j] = 1;  }  else if(program.Cells[program.i - 1][program.j + 1] == 0)  {  program.Cells[program.i - 1][program.j + 1] = 1;  }  else if(program.Cells[program.i + 1][program.j + 1] == 0)  {  program.Cells[program.i + 1][program.j + 1] = 1;  }  else if(program.Cells[program.i + 1][program.j] == 0)  {  program.Cells[program.i + 1][program.j] = 1;  }  }  else if((program.Cells[program.i][program.j - 1] > 2) && (program.Cells[program.i][program.j - 1] < 11)) // W... Done  {  if(program.Cells[program.i - 1][program.j] == 0)  {  program.Cells[program.i - 1][program.j] = 1;  }  else if(program.Cells[program.i - 1][program.j - 1] == 0)  {  program.Cells[program.i - 1][program.j - 1] = 1;  }  else if(program.Cells[program.i + 1][program.j - 1] == 0)  {  program.Cells[program.i + 1][program.j - 1] = 1;  }  else if(program.Cells[program.i + 1][program.j] == 0)  {  program.Cells[program.i + 1][program.j] = 1;  }  }  else if((program.Cells[program.i + 1][program.j - 1] > 2) && (program.Cells[program.i + 1][program.j - 1] < 11)) // SW... Done  {  if(program.Cells[program.i + 1][program.j] == 0)  {  program.Cells[program.i + 1][program.j] = 1;  }  else if(program.Cells[program.i][program.j - 1] == 0)  {  program.Cells[program.i][program.j - 1] = 1;  }  }  else if((program.Cells[program.i + 1][program.j] > 2) && (program.Cells[program.i + 1][program.j] < 11)) // S  {  if(program.Cells[program.i][program.j - 1] == 0)  {  program.Cells[program.i][program.j - 1] = 1;  }  else if(program.Cells[program.i][program.j + 1] == 0)  {  program.Cells[program.i][program.j + 1] = 1;  }  else if(program.Cells[program.i + 1][program.j - 1] == 0)  {  program.Cells[program.i + 1][program.j - 1] = 1;  }  else if(program.Cells[program.i + 1][program.j + 1] == 0)  {  program.Cells[program.i + 1][program.j + 1] = 1;  }  }  else if((program.Cells[program.i + 1][program.j + 1] > 2) && (program.Cells[program.i + 1][program.j + 1] < 11)) // SE... Done  {  if(program.Cells[program.i + 1][program.j] == 0)  {  program.Cells[program.i + 1][program.j] = 1;  }  else if(program.Cells[program.i][program.j + 1] == 0)  {  program.Cells[program.i][program.j + 1] = 1;  }  }  }      // Death for Fish  if(program.Cells[program.i][program.j] < 10)  {  program.Cells[program.i][program.j] += 1;  }  else  {  program.Cells[program.i][program.j] = 31; //Dead Fish  }    // Death by Shark  if((checkNeighbors.sharkNeighbor == 1) && (program.f < 0.1))  {  program.Cells[program.i][program.j] = 31; // Dead Fish  }  else if((checkNeighbors.sharkNeighbor == 2) && (program.f < 0.2))  {  program.Cells[program.i][program.j] = 31; // Dead Fish  }  else if((checkNeighbors.sharkNeighbor == 3) && (program.f < 0.3))  {  program.Cells[program.i][program.j] = 31; // Dead Fish  }  else if((checkNeighbors.sharkNeighbor == 4) && (program.f < 0.4))  {  program.Cells[program.i][program.j] = 31; // Dead Fish  }  else if((checkNeighbors.sharkNeighbor == 5) && (program.f < 0.5))  {  program.Cells[program.i][program.j] = 31; // Dead Fish  }  else if((checkNeighbors.sharkNeighbor == 6) && (program.f < 0.6))  {  program.Cells[program.i][program.j] = 31; // Dead Fish  }  else if((checkNeighbors.sharkNeighbor == 7) && (program.f < 0.7))  {  program.Cells[program.i][program.j] = 31; // Dead Fish  }  else if((checkNeighbors.sharkNeighbor == 8) && (program.f < 0.8))  {  program.Cells[program.i][program.j] = 31; // Dead Fish  }    // Death by overcrowding  if((checkNeighbors.fishNeighbor == 5) && (program.f < 0.05))  {  program.Cells[program.i][program.j] = 31; // Dead Fish  }  else if((checkNeighbors.fishNeighbor == 6) && (program.f < 0.1))  {  program.Cells[program.i][program.j] = 31; // Dead Fish  }  else if((checkNeighbors.fishNeighbor == 7) && (program.f < 0.15))  {  program.Cells[program.i][program.j] = 31; // Dead Fish  }  else if((checkNeighbors.fishNeighbor == 8) && (program.f < 0.20))  {  program.Cells[program.i][program.j] = 31; // Dead Fish  }    }      //////////////////////////////////////////////////////////////////////////////////////////////////////    // Meaning Shark  if((program.Cells[program.i][program.j] > 10) && (program.Cells[program.i][program.j] < 31))  {  // Breeding Rules  if(program.Cells[program.i][program.j] > 12 && program.Cells[program.i][program.j] < 31)  {  if((program.Cells[program.i - 1][program.j - 1] > 12) && (program.Cells[program.i - 1][program.j - 1] < 31)) // NW... DONE  {  if(program.Cells[program.i - 1][program.j] == 0)  {  program.Cells[program.i - 1][program.j] = 11;  }  else if(program.Cells[program.i][program.j - 1] == 0)  {  program.Cells[program.i][program.j - 1] = 11;  }  }  else if((program.Cells[program.i - 1][program.j] > 12) && (program.Cells[program.i - 1][program.j] < 31)) // N... Done  {  if(program.Cells[program.i - 1][program.j - 1] == 0)  {  program.Cells[program.i - 1][program.j - 1] = 11;  }  else if(program.Cells[program.i - 1][program.j + 1] == 0)  {  program.Cells[program.i - 1][program.j + 1] = 11;  }  else if(program.Cells[program.i][program.j - 1] == 0)  {  program.Cells[program.i][program.j - 1] = 11;  }  else if(program.Cells[program.i][program.j + 1] == 0)  {  program.Cells[program.i][program.j + 1] = 1;  }  }  else if((program.Cells[program.i - 1][program.j + 1] > 12) && (program.Cells[program.i - 1][program.j + 1] < 31)) // NE... Done  {  if(program.Cells[program.i - 1][program.j] == 0)  {  program.Cells[program.i - 1][program.j] = 11;  }  else if(program.Cells[program.i][program.j + 1] == 0)  {  program.Cells[program.i][program.j + 1] = 11;  }  }  else if((program.Cells[program.i][program.j + 1] > 12) && (program.Cells[program.i][program.j + 1] < 31)) // E... Done  {  if(program.Cells[program.i - 1][program.j] == 0)  {  program.Cells[program.i - 1][program.j] = 1;  }  else if(program.Cells[program.i - 1][program.j + 1] == 0)  {  program.Cells[program.i - 1][program.j + 1] = 11;  }  else if(program.Cells[program.i + 1][program.j + 1] == 0)  {  program.Cells[program.i + 1][program.j + 1] = 11;  }  else if(program.Cells[program.i + 1][program.j] == 0)  {  program.Cells[program.i + 1][program.j] = 11;  }  }  else if((program.Cells[program.i][program.j - 1] > 12) && (program.Cells[program.i][program.j - 1] < 31)) // W... Done  {  if(program.Cells[program.i - 1][program.j] == 0)  {  program.Cells[program.i - 1][program.j] = 11;  }  else if(program.Cells[program.i - 1][program.j - 1] == 0)  {  program.Cells[program.i - 1][program.j - 1] = 11;  }  else if(program.Cells[program.i + 1][program.j - 1] == 0)  {  program.Cells[program.i + 1][program.j - 1] = 11;  }  else if(program.Cells[program.i + 1][program.j] == 0)  {  program.Cells[program.i + 1][program.j] = 1;  }  }  else if((program.Cells[program.i + 1][program.j - 1] > 12) && (program.Cells[program.i + 1][program.j - 1] < 31)) // SW... Done  {  if(program.Cells[program.i + 1][program.j] == 0)  {  program.Cells[program.i + 1][program.j] = 11;  }  else if(program.Cells[program.i][program.j - 1] == 0)  {  program.Cells[program.i][program.j - 1] = 11;  }  }  else if((program.Cells[program.i + 1][program.j] > 12) && (program.Cells[program.i + 1][program.j] < 31)) // S  {  if(program.Cells[program.i][program.j - 1] == 0)  {  program.Cells[program.i][program.j - 1] = 11;  }  else if(program.Cells[program.i][program.j + 1] == 0)  {  program.Cells[program.i][program.j + 1] = 11;  }  else if(program.Cells[program.i + 1][program.j - 1] == 0)  {  program.Cells[program.i + 1][program.j - 1] = 11;  }  else if(program.Cells[program.i + 1][program.j + 1] == 0)  {  program.Cells[program.i + 1][program.j + 1] = 11;  }  }  else if((program.Cells[program.i + 1][program.j + 1] > 12) && (program.Cells[program.i + 1][program.j + 1] < 31)) // SE... Done  {  if(program.Cells[program.i + 1][program.j] == 0)  {  program.Cells[program.i + 1][program.j] = 11;  }  else if(program.Cells[program.i][program.j + 1] == 0)  {  program.Cells[program.i][program.j + 1] = 11;  }  }  }    // Death for Sharks  if(program.Cells[program.i][program.j] < 30)  {  program.Cells[program.i][program.j] += 1;  }  else  {  program.Cells[program.i][program.j] = 32; //Dead Shark  }    if(program.f < 0.0000)  {  program.Cells[program.i][program.j] = 32;  }    if((checkNeighbors.fishNeighbor == 0) && (program.f < 0.50))  {  program.Cells[program.i][program.j] = 32;  }    //Over Crowding  if((checkNeighbors.sharkNeighbor == 5) && (program.f < 0.30))  {  program.Cells[program.i][program.j] = 32;  }  else if((checkNeighbors.sharkNeighbor == 6) && (program.f < 0.45))  {  program.Cells[program.i][program.j] = 32;  }  else if((checkNeighbors.sharkNeighbor == 7) && (program.f < 0.60))  {  program.Cells[program.i][program.j] = 32;  }  else if((checkNeighbors.sharkNeighbor == 8) && (program.f < 0.75))  {  program.Cells[program.i][program.j] = 32;  }    }  ///////////////////////////////////////////////////////////////////////////////////////////////    if(program.Cells[program.i][program.j] > 30)  {  program.Cells[program.i][program.j] = 0;  }  }  } |

# 

# 7.19 checkNeighbours.c

|  |
| --- |
| public class checkNeighbors  {  static int fishNeighbor = 0;  static int sharkNeighbor = 0;    // fishNeighbor: 1 - 10  // sharkNeighbor: 11 -30  public checkNeighbors()  {  // NW  if ((program.Cells[program.i - 1][program.j - 1] < 11) && program.Cells[program.i - 1][program.j - 1] != 0)  {  fishNeighbor += 1;  }  else if((program.Cells[program.i - 1][program.j - 1] > 10) && (program.Cells[program.i - 1][program.j - 1] > 31))  {  sharkNeighbor += 1;  }    // N  if ((program.Cells[program.i - 1][program.j] < 11) && (program.Cells[program.i - 1][program.j] != 0))  {  fishNeighbor += 1;  }  else if((program.Cells[program.i - 1][program.j] > 10) && (program.Cells[program.i - 1][program.j] > 31))  {  sharkNeighbor += 1;  }    // NE  if ((program.Cells[program.i - 1][program.j + 1] < 11) && (program.Cells[program.i - 1][program.j + 1] != 0))  {  fishNeighbor += 1;  }  else if((program.Cells[program.i - 1][program.j + 1] > 10) && (program.Cells[program.i - 1][program.j + 1] > 31))  {  sharkNeighbor += 1;  }    // W  if ((program.Cells[program.i][program.j - 1] < 11) && program.Cells[program.i][program.j - 1] != 0)  {  fishNeighbor += 1;  }  else if((program.Cells[program.i][program.j - 1] > 10) && (program.Cells[program.i][program.j - 1] > 31))  {  sharkNeighbor += 1;  }    // E  if ((program.Cells[program.i][program.j + 1] < 11) && program.Cells[program.i][program.j + 1] != 0)  {  fishNeighbor += 1;  }  else if((program.Cells[program.i][program.j + 1] > 10) && (program.Cells[program.i][program.j + 1] > 31))  {  sharkNeighbor += 1;  }    // SW  if ((program.Cells[program.i + 1][program.j - 1] < 11) && program.Cells[program.i + 1][program.j - 1] != 0)  {  fishNeighbor += 1;  }  else if((program.Cells[program.i + 1][program.j - 1] > 10) && (program.Cells[program.i + 1][program.j - 1] > 31))  {  sharkNeighbor += 1;  }    // S  if ((program.Cells[program.i + 1][program.j] < 11) && program.Cells[program.i + 1][program.j] != 0)  {  fishNeighbor += 1;  }  else if((program.Cells[program.i + 1][program.j] > 10) && (program.Cells[program.i + 1][program.j] > 31))  {  sharkNeighbor += 1;  }    // SE  if ((program.Cells[program.i + 1][program.j + 1] < 11) && program.Cells[program.i + 1][program.j + 1] != 0)  {  fishNeighbor += 1;  }  else if((program.Cells[program.i + 1][program.j + 1] > 10) && (program.Cells[program.i + 1][program.j + 1] > 31))  {  sharkNeighbor += 1;  }  }  } |