Sprawozdanie z przedmiotu
Programowanie Równoległe i Rozproszone

Gra w życie

Wykonał:

Imię i nazwisko	Magdalena Paszko
Nr indeksu	76024
Grupa	L2

Prowadzący : dr inż. Krzysztof Szerszeń

1. Wprowadzanie

Gra w życie została opracowana w 1970 roku przez brytyjskiego matematyka Johna Conwaya. Jest to jeden z pierwszych i najpopularniejszych przykładów automatu komórkowego. Toczy się ona na planszy podzielonej na komórki. Każda z komórek ma ośmiu sąsiadów oraz może znajdować się w jednym z dwóch stanów. Każda z nich może być włączona (żywa), bądź wyłączona (martwa). Stany zmieniają się co turę w każdej komórce. Stan komórki zależy od liczby jej żywych sąsiadów. W tej grze nie występuje gracz jako osoba wykonująca ruchy. Jedyna jego ingerencja ogranicza się do ustalenia początkowego stanu komórek.

2. Reguły gry:

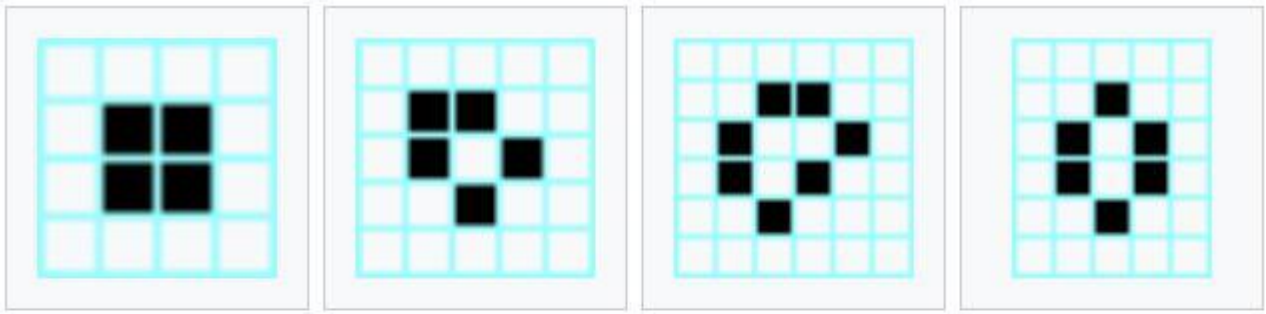
Reguły określają kiedy komórka jest włączona, a kiedy wyłączona.

- Komórka jest włączona zawsze na wolnym polu, które jest otoczone przez 3 zajęte pola

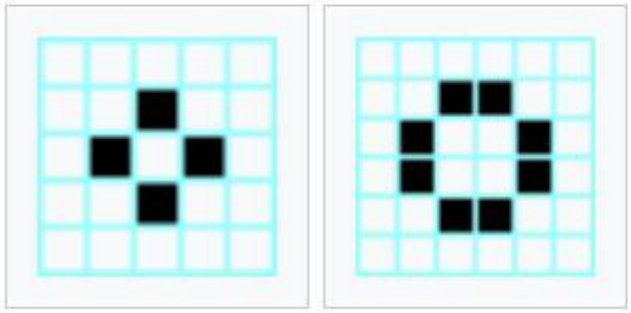
- Komórka zostaje wyłączona, gdy ma w sąsiedztwie mniej niż 2 sąsiadów, lub gdy jest ich więcej niż 3

3. Przykładowe struktury:

a) Niezmienne :



Klocek (blok) | Łódź | Bochenek | Kryształ (ang. *beehive* – ul)



Koniczynka (ang. *tub* – kadź) | Staw (ang. *pond*)

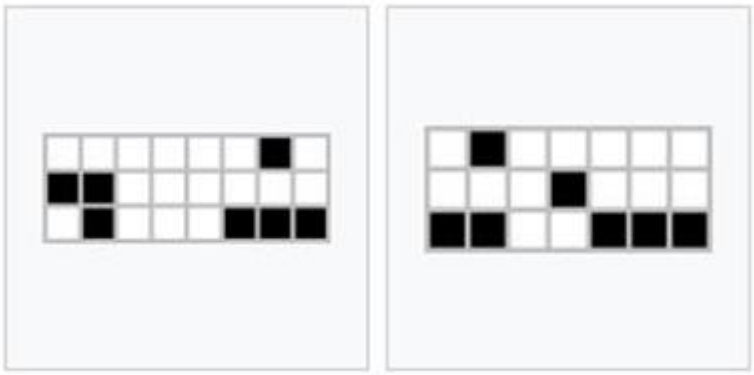
Niezmienne struktury pozostają włączone bez względu na iterację wykonywania programu.
Zwykle pojawiają się na końcu wykonywanego programu – w końcowych iteracjach.
W dalszej części sprawozdania pokażę taką strukturę w stworzonym przeze mnie przykładzie.

b) Oscylatory:

Blinker (światła uliczne) Żabka Krokodyl

Zmieniają się okresowo, a co pewną iterację wracają do stanu początkowego.

c) Niestałe:



Diehard – umiera po 130 krokach; ; L≈15.

Żołądź – stabilizuje się po 5206 krokach; L≈700.

Są to struktury, które zmieniają się i nigdy nie wracają do swojego stanu pierwotnego, a w efekcie giną po określonej liczbie kroków.

4. Implementacja algorytmu:

```
int main(int argc, char **argv)
{

    int plansza[64] =
        {0, 0, 0, 0, 0, 0, 0, 1,
         0, 1, 0, 0, 0, 1, 0, 0,
         0, 0, 0, 1, 0, 0, 0, 0,
         0, 1, 0, 1, 0, 0, 0, 1,
         0, 1, 0, 1, 1, 0, 0, 0,
         0, 1, 0, 0, 0, 0, 0, 1,
         0, 0, 1, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 1, 0,};

    int num_procs;
    int ID, j;
    int ktoraIteracja = 0;
    int liczbaIteracji;

    if (argc == 1) liczbaIteracji = ILEITERACJI;
    else if (argc == 2) num_iterations = atoi(argv[1]);
    else {
        printf("error\n");
        exit(1);
    }
    MPI_Status stat;

    if(MPI_Init(&argc, &argv) != MPI_SUCCESS) printf("MPI_Init error\n");

    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
    MPI_Comm_rank(MPI_COMM_WORLD, &ID);

    assert(SZEROKOSC % num_procs == 0);
```

Tablica „plansza” reprezentuje planszę do gry, a każda jej wartość to komórka. Komórki jest włączona, gdy jej wartość wynosi 1, a wyłączona dla wartości 0.
Poniżej definiujemy zmienne na które będą potrzebne do ustawienia środowiska oraz definiujemy środowisko MPI.

```
int *arr = (int *)malloc(SZEROKOSC * ((SZEROKOSC / num_procs) + 2) * sizeof(int));
for (ktoraIteracja = 0; ktoraIteracja < liczbaIteracji; ktoraIteracja++) {
    j = SZEROKOSC;
    for (int i = ID * (ROZMIAR / num_procs); i < (ID + 1) * (ROZMIAR / num_procs); i++) {
        arr[j] = plansza[i];
        j++;
    }
    if (num_procs != 1) {
        int incoming_1[SZEROKOSC];
        int incoming_2[SZEROKOSC];
        int send_1[SZEROKOSC];
        int send_2[SZEROKOSC];
        if (ID % 2 == 0) {
            for (int i = 0; i < SZEROKOSC; i++)
                send_1[i] = arr[i + SZEROKOSC];

            MPI_Ssend(&send_1, SZEROKOSC, MPI_INT, mod(ID - 1, num_procs), 1, MPI_COMM_WORLD);

            for (int i = 0; i < SZEROKOSC; i++)
                send_2[i] = arr[(SZEROKOSC * (SZEROKOSC / num_procs)) + i];

            MPI_Ssend(&send_2, SZEROKOSC, MPI_INT, mod(ID + 1, num_procs), 1, MPI_COMM_WORLD);
        } else {
            MPI_Recv(&incoming_2, SZEROKOSC, MPI_INT, mod(ID + 1, num_procs), 1, MPI_COMM_WORLD, &stat);
            MPI_Recv(&incoming_1, SZEROKOSC, MPI_INT, mod(ID - 1, num_procs), 1, MPI_COMM_WORLD, &stat);
        }
        if (ID % 2 == 0) {
            MPI_Recv(&incoming_2, SZEROKOSC, MPI_INT, mod(ID + 1, num_procs), 1, MPI_COMM_WORLD, &stat);
            MPI_Recv(&incoming_1, SZEROKOSC, MPI_INT, mod(ID - 1, num_procs), 1, MPI_COMM_WORLD, &stat);
        } else {
            for (int i = 0; i < SZEROKOSC; i++)
                send_1[i] = arr[i + SZEROKOSC];

            MPI_Ssend(&send_1, SZEROKOSC, MPI_INT, mod(ID - 1, num_procs), 1, MPI_COMM_WORLD);

            for (int i = 0; i < SZEROKOSC; i++)
                send_2[i] = arr[(SZEROKOSC * (SZEROKOSC / num_procs)) + i];

            MPI_Ssend(&send_2, SZEROKOSC, MPI_INT, mod(ID + 1, num_procs), 1, MPI_COMM_WORLD);
        }
        for (int i = 0; i < SZEROKOSC; i++) {
            arr[i] = incoming_1[i];
            arr[(SZEROKOSC * ((SZEROKOSC / num_procs) + 1)) + i] = incoming_2[i];
        }
    } else {
        for (int i = 0; i < SZEROKOSC; i++)
            arr[i + ROZMIAR + SZEROKOSC] = plansza[i];

        for (int i = ROZMIAR; i < ROZMIAR + SZEROKOSC; i++)
            arr[i - ROZMIAR] = plansza[i - SZEROKOSC];
    }
```

W dalszej części funkcji main definiujemy środowisko w zależności od ilości procesów przeznaczonych na wykonanie zadania. Każdy proces przyjmuje na siebie odpowiednia ilość wierszy. Przykładowo dając 4 procesy na zadanie z 8 wierszami każdy z nich zajmuje się dwoma wierszami. Procesy oprócz swoich wierszy widzą ostatni wiersz poprzedzającego procesu oraz pierwszy wiersz następnego procesu. Główny proces służy jedynie do rozdzielenia wierszy, przechowywania danych oraz późniejszej aktualizacji naszej planszy.

```
int * final = (int *)malloc(SZEROKOSC * ((SZEROKOSC / num_procs)) * sizeof(int));

for (int k = SZEROKOSC; k < SZEROKOSC * ((SZEROKOSC / num_procs) + 1); k++) {
    int total_rows = SZEROKOSC * (SZEROKOSC / num_procs) + 2;
    int r = k / SZEROKOSC;
    int c = k % SZEROKOSC;
    int prev_r = mod(r - 1, total_rows);
    int prev_c = mod(c - 1, SZEROKOSC);
    int next_r = mod(r + 1, total_rows);
    int next_c = mod(c + 1, SZEROKOSC);

    int count = arr[prev_r * SZEROKOSC + prev_c] + arr[prev_r * SZEROKOSC + c] + arr[prev_r * SZEROKOSC + next_c]
    + arr[r * SZEROKOSC + prev_c] + arr[r * SZEROKOSC + next_c] + arr[next_r * SZEROKOSC + prev_c] + arr[next_r * SZEROKOSC + c] + arr[next_r * SZEROKOSC + next_c];
    if (arr[k] == 1) {
        if (count < 2)
            final[k - SZEROKOSC] = 0;
        else if (count > 3)
            final[k - SZEROKOSC] = 0;
        else
            final[k - SZEROKOSC] = 1;
    } else {
        if (count == 3)
            final[k - SZEROKOSC] = 1;
        else
            final[k - SZEROKOSC] = 0;
    }
}
j = 0;
for (int i = ID * (ROZMIAR / num_procs); i < (ID + 1) * (ROZMIAR / num_procs); i++) {
    plansza[i] = final[j];
    j++;
}
MPI_Gather(final, SZEROKOSC * (SZEROKOSC / num_procs), MPI_INT, &plansza, SZEROKOSC * (SZEROKOSC / num_procs), MPI_INT, 0, MPI_COMM_WORLD);
```

Teraz ustawiamy na naszej planszy sąsiadów dla komórek zgodnie z zasadami gry, czyli włączamy nowe komórki oraz wyłączamy stare nie spełniające wymagań istnienia. Pozycja jest liczona wykorzystując funkcję mod, która sprawdza zakres tablicy i odpowiednio przydziela indeks.

```
if (ID == 0) {
    printf("\nNr iteracji: %d, Gra w życie:\n", ktoraIteracja);
    for (j = 0; j < ROZMIAR; j++) {
        if (j % SZEROKOSC == 0)
            printf("\n");

        printf("%d  ", plansza[j]);
    }
    printf("\n");
}
```

Na koniec zostaje nam ustawienie na planszy komórek zgodnie z ich stanem. Wszystko to dzieje się zgodnie z wykonywaną iteracją. Ukazaniem aktualnej planszy zajmuje się główny proces.

Wynik:

```
magda@magda :~/Documents$ mpicc main.c -o main
magda@magda :~/Documents$ mpirun -np 8 main

Nr iteracji: 0, Gra w życie:

0  0  0  0  0  0  1  0
0  0  0  0  0  0  0  0
1  0  0  0  1  0  0  0
1  0  0  1  0  0  0  0
0  1  0  1  1  0  0  0
1  1  0  1  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0

Nr iteracji: 1, Gra w życie:

0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
1  1  1  1  0  0  0  0
0  1  0  1  1  0  0  0
1  1  0  1  1  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
```

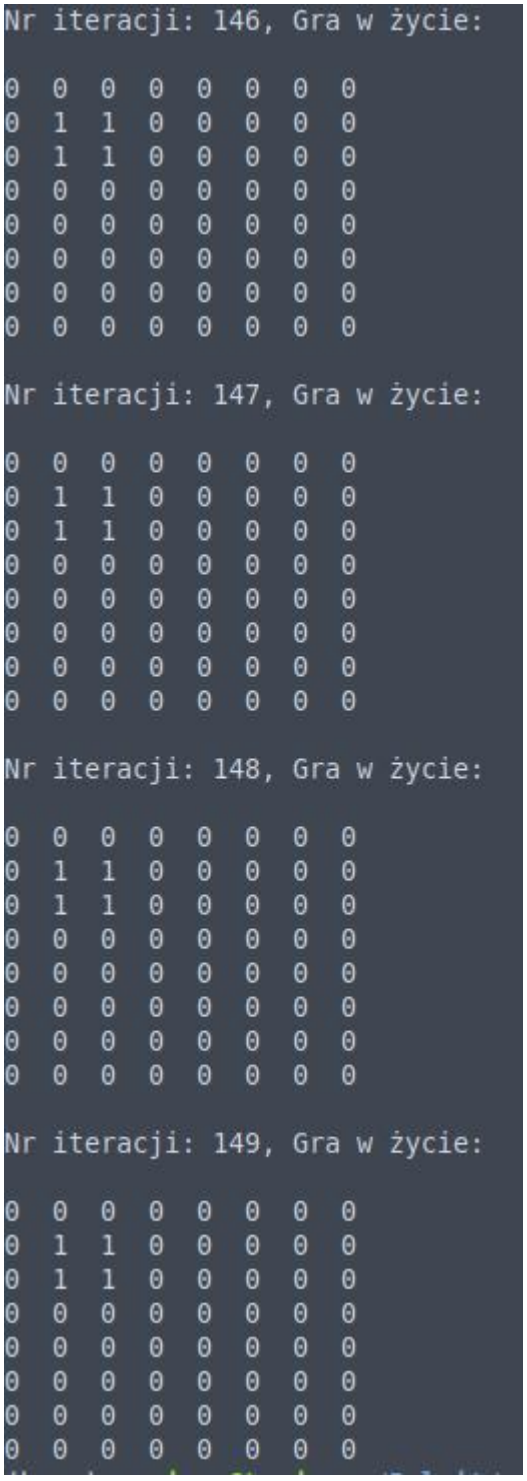
Kompilujemy nasz program – mpicc main.c -o main ,a następnie uruchamiamy go przeznaczając pewną ilość procesów, w tym przypadku 8: mpirun -np 8 main.

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

```
0  0  0  0  0  0  1  0
0  0  0  0  0  0  0  0
1  0  0  0  1  0  0  0
1  0  0  1  0  0  0  0
0  1  0  1  1  0  0  0
1  1  0  1  0  0  0  0
0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0
```

Po lewej na białym tle widać początkowy stan planszy, po prawej pierwsza wykonana iteracja. Jak widać przykładowo dla przedostatniej komórki (druga z prawej od dołu) została wyłączona, ponieważ jest samotna (nie ma nikogo w sąsiedztwie).

Każda kolejna iteracja jest wykonywana, aż do końca, czyli ostatniej iteracji określonej przez nas.



Jak widać powstała na koniec struktura niezmienna - Klocek (blok). Występuje ona od wielu iteracji. W „Grze w życie” możemy definiować wiele takich struktur i obserwować ich żywot.