

SPRAWOZDANIE

CEL PROJEKTU

Celem projektu było zaimplementowanie w języku c programu obliczającego kolejne etapy mrówki Langtona i zapisującego je do plików lub na standardowe wyjście jeśli nie zostanie podana nazwa pliku.

Ideą programu jest symulowanie ruchu mrówki na dwuwymiarowej siatce komórek, co może prowadzić do utworzenia skomplikowanych i interesujących wzorów. Działanie mrówki Langtona jest oparte na pewnych zasadach:

1. Siatka komórek:

- Mrówka chodzi po dwuwymiarowej siatce, której każda komórka może mieć jeden z dwóch możliwych stanów: biały lub czarny.

2. Mrówka:

- Mrówka może poruszać się w jednym z 4-ech kierunków (góra, dół, lewo, prawo). Początkowy kierunek mrówki podawany jest przez użytkownika.
- Początkową pozycją mrówki w przypadku opisywanego programu jest środkowa komórka siatki (w przypadku parzystej długości boków – najbliższa komórka na lewo i w górę od środka) lub ewentualnie pozycja odczytana z podanego przy wywołaniu pliku.

3. Zasady ruchu mrówki:

- Jeśli mrówka znajduje się w białej komórce, wykonuje: obrót o 90 stopni w prawo, zmienia kolor komórki na przeciwny i przesuwa się o jedną komórkę do przodu
- Jeśli mrówka znajduje się w czarnej komórce, wykonuje: obrót o 90 stopni w lewo, zmienia kolor komórki na przeciwny i przesuwa się o jedną komórkę do przodu

4. Powtarzanie kroków:

- Ruch mrówki powtarza się zgodnie z zasadami tyle razy, ile iteracji podał użytkownik
- W przypadku, gdy mrówka dojdzie do brzegu siatki, wychodzi z jej przeciwnej strony (górny brzeg – wchodzi od dołu siatki, prawy brzeg – wychodzi od lewej i odwrotnie)

WYWOŁANIE PROGRAMU

Program należy na początku skompilować poleceniem „make”.

Następnie uruchamiamy go używając nazwy pliku wykonywalnego „program” oraz podanie następujących parametrów:

- -m <liczba wierszy>
- -n <liczba kolumn>
- -i <liczba iteracji>
- -o <przedrostek plików wynikowych>
- -d <początkowy kierunek mrówki> (podajemy jako N [góra], S [dół], E [prawo] lub W [lewo])
- -l <nazwa pliku z siatką, „czarnymi polami” i aktualną pozycją mrówki> (parametr opcjonalny)
- -p <procentowe (liczba zmiennoprzecinkowa) wypełnienie planszy „czarnymi” polami> (parametr opcjonalny)

```
~/mrowka1$ ./program -m 51 -n 101 -i 11000 -o plik -d N
~/mrowka1$ ./program -m 51 -n 101 -i 11000 -o plik2 -d N -l plik_11000
~/mrowka1$ ./program -m 51 -n 101 -i 11000 -o plik3 -d N -p 50
```

Rys. 1.1 – 1.3 (przykładowe wywołanie programu z wypisem do pliku)

```
~/mrowka1$ ./program -m 25 -n 51 -i 10 -d N -p 20_
```

Rys. 2.1 (przykładowe wywołanie programu z wypisem na stdout)

OPIS ROZWIĄZANIA

Program składa się z głównego pliku main.c oraz trzech modułów: inicjacja, ruch, wypisywanie.

Używane są dwie struktury:

1. mrowka, w której:

- d – kierunek mrówki (0 – góra, 1 – prawo, 2 – dół, 3 – lewo),
- x – indeks kolumny, w której znajduje się mrówka,
- y – indeks wiersza, w którym znajduje się mrówka.

```
typedef struct{
    int d;
    int x; //n ant_location
    int y; //m ant_location
} mrowka;
```

Rys. 3.1

2. pole_m, w której:

- **s – dwuwymiarowa tablica, której każdy element reprezentuje komórkę siatki i przyjmuje wartość: 1 jeśli kolor komórki jest czarny, 0 jeśli biały,
- m - liczba wierszy siatki,
- n – liczba kolumn siatki,
- p – procentowe zapełnienie siatki.

```
typedef struct{
    int **s;
    int m;
    int n;
    double p;
} pole_m;
```

Rys. 3.2

main.c

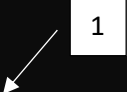
Główny plik zawiera funkcję „main”, która obsługuje argumenty linii poleceń, zapewnia obsługę błędów oraz wywołuje odpowiednie funkcje.

Do przetwarzania argumentów wywołania wykorzystywana jest funkcja getopt (Rys. 3.3). W przypadku podania przez użytkownika niepoprawnego parametru, program wypisuje komunikat o błędzie i kończy działanie. (1)

```

//Przetwarzanie parametrów
while ((opt=getopt(argc, argv, "m:n:i:o:d:l:p:")) != -1){
    switch (opt){
        case 'm':
            pole.m=atoi(optarg);
            break;
        case 'n':
            pole.n=atoi(optarg);
            break;
        case 'i':
            it=atoi(optarg);
            break;
        case 'o':
            o=optarg;
            break;
        case 'd':
            pom_d=optarg[0];
            break;
        case 'l':
            l=optarg;
            break;
        case 'p':
            pole.p=atof(optarg);
            spr=1;
            break;
        default:
            fprintf(stderr, "Użycie ./a.out -m <wiersze> -n <kolumn> -i <iteracje>
-o <plik> -d <kierunek> -l(opcjonalny) <mapa do wczytania> -r(opcjonalny) <procentowe zapełnienie>\n");
            return EXIT_FAILURE;
    }
}

```



Rys. 3.3

Program obsługuje błędy takie jak (Rys. 3.4): brak/niepoprawna liczba wierszy lub kolumn (1), brak/niepoprawna liczba iteracji (2), niepoprawna wartość procentowego zapełnienia siatki (3), podanie dwóch obu dodatkowych funkcji na raz (4) oraz nieprawidłowo podany kierunek początkowy mrówki (5).

```

//Obsługa błędów
if (pole.m<=0 || pole.n<=0){
    fprintf(stderr, "Należy podać dodatnią liczbę wierszy i kolumn.\n");
    return 1;
}
if (it<=0){
    fprintf(stderr, "Należy podać dodatnią liczbę iteracji.\n");
    return 1;
}
if (spr==1 && (pole.p<=0 || pole.p>100)){
    fprintf(stderr, "Niepoprawna wartość procentowego wypełnienia planszy.\n");
    return 1;
}
if (ll=NULL && pole.p>0){
    fprintf(stderr, "Proszę wybrać jedną z dwóch dodatkowych opcji.\n");
    return 1;
}
switch (pom_d){
    case 'N':
        ant.d=0;
        break;
    case 'E':
        ant.d=1;
        break;
    case 'S':
        ant.d=2;
        break;
    case 'W':
        ant.d=3;
        break;
    default:
        fprintf(stderr, "Podaj kierunek mrówki jako N E S lub W.\n");
        return 1;
}

```

Rys. 3.4

Alokowana jest pamięć dla tablicy dwuwymiarowej reprezentującej siatkę. (Rys. 3.5)

```

//Inicjacja siatki
pole.s=malloc(pole.m*sizeof(int*));
for (i=0; i<pole.m; i++)
    pole.s[i]=malloc(pole.n*sizeof(int));

```

Rys. 3.5

Program sprawdza, czy użytkownik podał dodatkową opcję wczytania mapy z pliku, w przeciwnym przypadku ustawia pozycję mrówki jako środkową komórkę i wypełnia tablicę zerami reprezentującymi biały kolor komórek oraz sprawdza czy podane zostało przez użytkownika procentowe wypełnienie planszy. (Rys. 3.6)

```

//Obsługa wczytywania z pliku
if (l!=NULL){
    pom=wczytaj(&ant, &pole, l);
    if (pom==1)
        return 1;
}else{
    ant.y=(pole.m-1)/2;
    ant.x=(pole.n-1)/2;
    for (i=0; i<pole.m; i++)
        for (j=0; j<pole.n; j++)
            pole.s[i][j]=0;

    if (pole.p>0){
        los(&pole);
    }
}
}

```

Rys. 3.6

Na koniec wywoływane są funkcje obliczające i wypisujące kolejne etapy mrówki Langtona. (Rys. 3.7)

```

//Obliczanie i wypisywanie etapów
pom=wypisz(&ant, &pole, 0, o);
if (pom==1)
    return 1;
for (i=1; i<=it; i++){
    move(&ant, &pole);
    pom=wypisz(&ant, &pole, i, o);
    if (pom==1)
        return 1;
}

return 0;

```

Rys. 3.7

inicjacja

Moduł inicjacja zawiera dwie funkcje obsługujące dodatkowe opcje podane przez użytkownika:

- funkcja „wczytaj”
Używa struktury mrowka i pole_m. Jest ona odpowiedzialna za wczytanie mapy z naniesionymi już „czarnymi” polami i aktualną pozycją mrówki z pliku podanego przez użytkownika jako parametr -l.

Zaczyna wczytywanie, gdy napotka znak „r” (1) oraz pomija zewnętrzne krawędzie siatki (2). Funkcja w zależności od napotkanego znaku, przypisuje odpowiadającemu komórce elementowi tablicy s ze struktury pole_m wartość 1 dla pola czarnego lub 0 dla białego (3) oraz gdy napotka znak reprezentujący mrówkę, zapisuje jej pozycję do struktury mrowka (4).

Funkcja obsługuje też błędy: jeśli napotka znak reprezentujący krawędź, wypisuje komunikat, że wartości podane dla parametrów m i n nie zgadzają się z rozmiarem siatki z pliku (5); nieprawidłowy znak w siatce (6); brak pozycji mrówki (7); błąd otwarcia pliku (8).

```

if (in!=NULL){
    while ((z=fgetc(in)) !=WEOF){
        while (z==L'\n')
            z=fgetc(in);
        if (nn==0 && mm==0){
            while (z!=L'r')
                z=fgetc(in); 1
        }
        if (z==WEOF)
            break;
        if (nn==pole->n+2){
            mm++;
            nn=0;
        }
        if (mm==pole->m+2)
            break;
        2 if (mm==0 || nn==0 || nn==pole->n+1 || mm==pole->m+1){
            nn++;
            continue;
        }else if (z==L'█'){
            pole->s[mm-1][nn-1]=1;
        }else if (z==L'▲' || z==L'⬢' || z==L'▼' || z==L'⬢'){
            ant->x=nn-1;
            ant->y=mm-1;
            pole->s[mm-1][nn-1]=1;
        }else if (z==L'⬢' || z==L'⬢' || z==L'⬢' || z==L'⬢'){
            ant->x=nn-1;
            ant->y=mm-1;
            pole->s[mm-1][nn-1]=0;
        }else if (z==L' '){
            pole->s[mm-1][nn-1]=0;
        }else if (z==L'r' || z==L'-' || z==L'_' || z==L'|' || z==L'.' || z==L','){
            fprintf(stderr, "Podane wartości m i n nie zgadzają się z wczytanym plikiem.\n"); 5
            return 1;
        }else{
            fprintf(stderr, "Plik zawiera nieprawidłowy znak.\n"); 6
            return 1;
        }
        nn++;
    }
}

```

Rys. 4.1

```

    if (ant->x==--1 || ant->y==--1){
        fprintf(stderr, "Nie mogę odczytać pozycji mrówki.\n"); 7
        return 1;
    }
    }else{
        fprintf(stderr, "Błąd otwarcia pliku %s\n", 1); 8
        return 1;
    }
    return 0;
}

```

Rys. 4.2

- funkcja „los”

Używa struktury pole_m. Jest odpowiedzialna za wygenerowanie mapy z losowo ustawionymi „czarnymi” polami według procentowego zapełnienia planszy podanego przez użytkownika.

Oblicza ona ilość czarnych komórek według podanej wartości procentowej (1), a następnie losuje tyle numerów komórek (2) i przypisuje wartość 1 odpowiadającym im elementom dwuwymiarowej tablicy s (3).

```

int los(pole_m * pole){
    int i, j;
    int x=pole->m*pole->n*pole->p/100;
    int wylosowane[x];
    int k=pole->m*pole->n;
    int m,n;
    srand(time(NULL));
    for (i=0; i<x; i++){
        wylosowane[i]=rand() % k;
        for (j=0; j<i; j++)
            if (wylosowane[i] == wylosowane[j]){
                i--;
                break;
            }
    }
    for (i=0; i<x; i++){
        m=wylosowane[i]/pole->n;
        n=wylosowane[i]%pole->n;
        pole->s[m][n]=1;
    }

    return 0;
}

```

Rys. 4.3

ruch

Moduł zawiera jedną funkcję odpowiedzialną za ruch mrówki:

- funkcja „move”

Funkcja używa dwóch struktur mrowka i pole_m. Oblicza ona kolejne etapy ruchu mrówki.

Jeżeli kolor komórki, w której mrówka aktualnie się znajduje to biały (element tablicy s równy 0), to zmienia ona kierunek mrówki o jeden obrót w prawo i zmienia kolor pola na czarny(1). W przeciwnym przypadku zmienia kierunek mrówki o jeden obrót w lewo i zmienia kolor pola na biały (2). Następnie w zależności od kierunku mrówki, jej pozycja przesuwana jest o jeden do przodu (3).

```

int move(mrowka *a, pole_m * p){
    int d=a->d;
    int t;
    t=p->s[a->y][a->x];
    if (t==0){
        a->d++;
        if (a->d>3)
            a->d-=4;
        p->s[a->y][a->x]=1;
    }else{
        a->d--;
        if (a->d<0)
            a->d+=4;
        p->s[a->y][a->x]=0;
    }
    switch (a->d){
        case 0:
            a->y--;
            if (a->y<0)
                a->y+=p->m;
            break;
        case 1:
            a->x=(a->x+1)%p->n;
            break;
        case 2:
            a->y=(a->y+1)%p->m;
            break;
        case 3:
            a->x--;
            if (a->x<0)
                a->x+=p->n;
            break;
    }
    return 0;
}

```

Rys. 5.1

wypisywanie

Moduł zawiera dwie funkcje odpowiedzialne za wypisywanie etapu mrówki do pliku lub na stdout:

- funkcja „mr”
Używa obu struktur. Zwraca odpowiedni znak reprezentujący mrówkę w zależności od koloru pola i aktualnego kierunku mrówki (Rys. 6.1)


```

char *mr( mrowka * ant, pole_m * pole){
    int d=ant->d;
    int t=pole->s[ant->y][ant->x];
    if (d==0)
        if (t==1)
            return "▲";
        else
            return "▢";
    if (d==2)
        if (t==1)
            return "▼";
        else
            return "▢";
    if (d==1)
        if (t==1)
            return "▢";
        else
            return "▢";
    if (d==3)
        if (t==1)
            return "▢";
        else
            return "▢";
}

```

Rys. 6.1

- funkcja „wypisz”

Używa obu struktur.

Na początku, jeśli podano przedrostek plików wynikowych, tworzy ona nazwę pliku (w formacie „namefile_nriteracji”), do którego wypisywać będzie dany etap, w przeciwnym przypadku siatka wypisana zostanie na stdout (Rys. 7.1).

Następnie, w zależności od pozycji w której się znajduje oraz koloru komórki reprezentowanego przez wartości tablicy s (0 – biały, 1 – czarny), wypisuje odpowiedni znak (Rys. 7.2).

```

if (o!=NULL){
    for (i=10; (nr/i)>=1; i*=10)
        len++;
    char numer[len];
    len+=strlen(o)+1;
    char nazwa[len];
    sprintf(numer, "%d", nr);
    strcpy(nazwa, "");
    strcat(nazwa, o);
    strcat(nazwa, "_");
    strcat(nazwa, numer);
    out = fopen(nazwa,"w");
}else
    out = stdout;

if (out==NULL){
    fprintf(stderr, "Nie moze pisac do %s_%d\n", o, nr);
    return 1;
}

```

Rys. 7.1

```

for(i=0; i<m+2; i++){
    for(j=0; j<n+2; j++){
        if (i-1==ant->y && j-1==ant->x)
            fprintf(out,"%s", mr(ant,pole));
        else if (i>0 && j>0 && i-1<m && j-1<n && pole->s[i-1][j-1]==1)
            fprintf(out, "█");
        else if (i==0 && j==0)
            fprintf(out, "┐");
        else if (i==0 && j==n+1)
            fprintf(out, "┌");
        else if (i==m+1 && j==0)
            fprintf(out, "└");
        else if (i==m+1 && j==n+1)
            fprintf(out, "┘");
        else if (j==0 || j==n+1)
            fprintf(out, "|");
        else if (i==0 || i==m+1)
            fprintf(out, "-");
        else if (i>0 && j>0 && i-1<m && j-1<n && pole->s[i-1][j-1]==0)
            fprintf(out, " ");
        fprintf(out, "\n");
    }
}

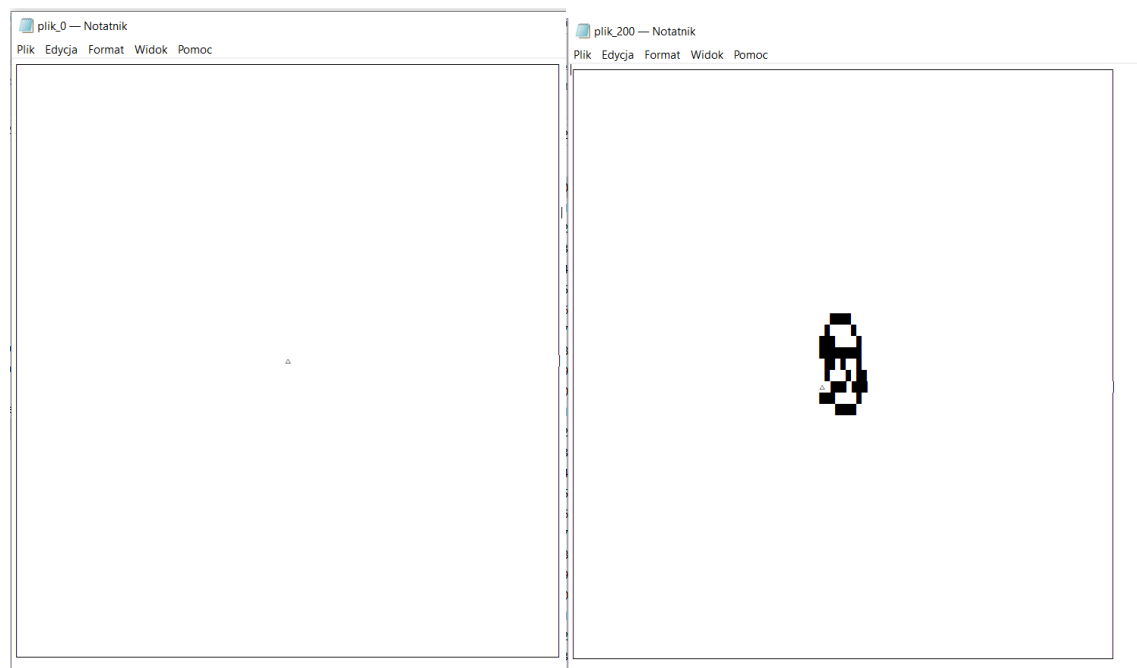
```

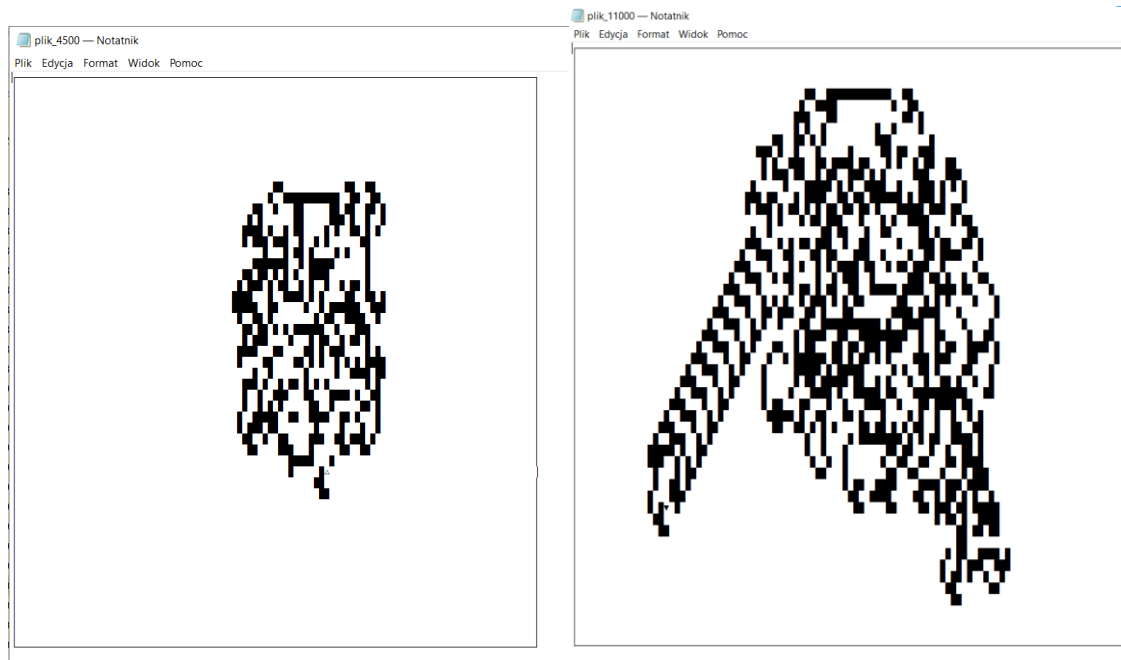
Rys. 7.2

PRZYKŁADOWE DZIAŁANIE PROGRAMU

- Bez dodatkowych opcji

```
~/mrowka1$ ./program -m 51 -n 101 -i 11000 -o plik -d N
```

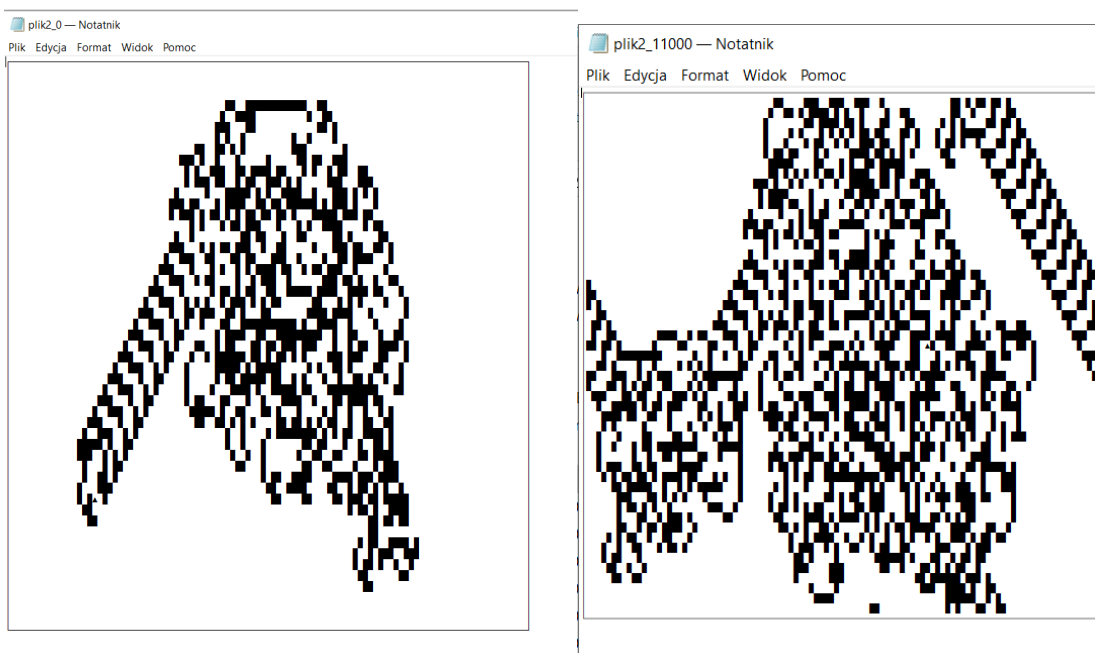




Rys. 8.1 – 8.4 (plik_0, plik_200, plik_4500, plik_11000)

- z opcją -l plik_11000

```
~/mrowka1$ ./program -m 51 -n 101 -i 11000 -o plik2 -d N -l plik_11000
```



Rys. 8.5 – 8.6 (plik2_0, plik2_11000)

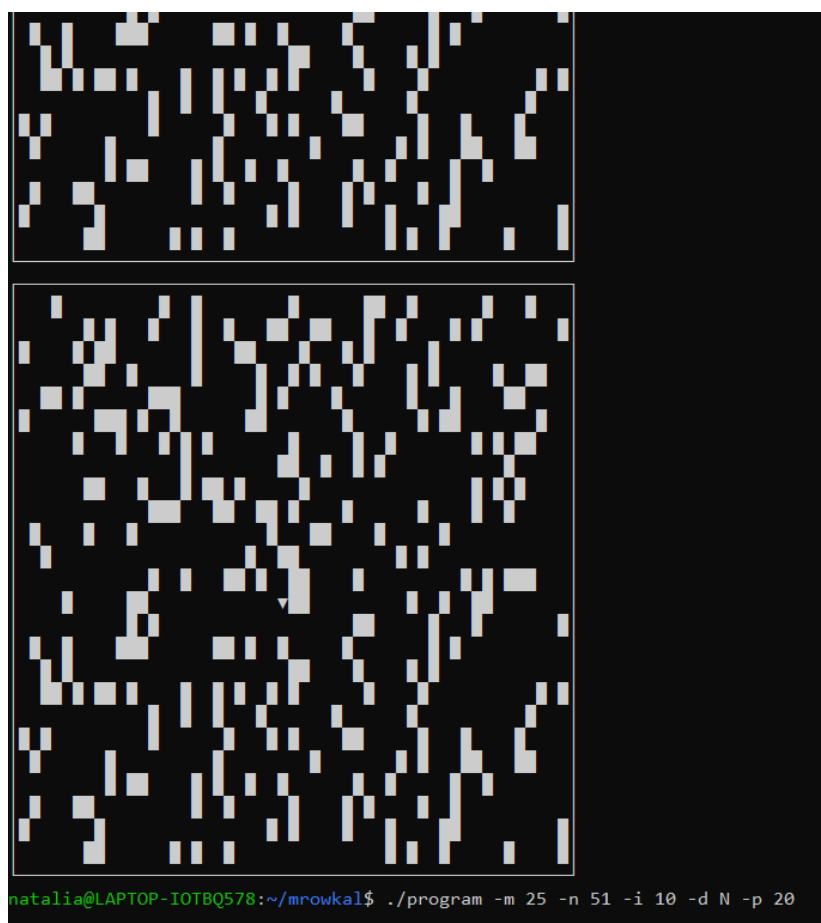
- z opcją -p 50

```
~/mrowka1$ ./program -m 51 -n 101 -i 11000 -o plik3 -d N -p 50
```



Rys. 8.7 – 8.8 (plik3_0, plik3_11000)

- wypisanie na stdout



Rys. 8.9

WNIOSKI

Cele projektu zostały osiągnięte. Program oblicza kolejne etapy mrówki Langtona zgodnie z przyjętymi zasadami i wypisuje je do plików w formacie „namefile_nriteracji” lub na stdout. Umożliwia także opcjonalne wygenerowanie w programie mapy z naniesionymi już „czarnymi” polami i aktualną pozycją mrówki oraz opcjonalne wygenerowanie w programie mapy z losowo ustawionymi „czarnymi” polami/ przeszkodami według procentowego zapełnienia planszy podanego przez użytkownika. Program można wykorzystać do tworzenia ciekawych wzorów. Już przy 11000 iteracji możemy zauważyć charakterystyczny dla mrówki Langtona wzór. Dzięki dodatkowym opcjom powstające obrazy mogą być jeszcze bardziej skomplikowane.