19.01.2024r.

Sprawozdanie

Bartłomiej Ciupak
Tomasz Rogalski

1. *Wstęp*
   Mrówka Langtona to prosty automat komórkowy wymyślony przez Chrisa Langtona. Mrówka chodzi po dwuwymiarowej siatce. Obowiązują dwie zasady ruchu i jej wpływu na przestrzeń:
   - jeśli mrówka wejdzie na białą komórkę przestrzeni to zmienia jej kolor na czarny i pod kątem prostym skręca w prawo do następnej komórki,
   - jeśli mrówka wejdzie na czarną komórkę przestrzeni to zmienia jej kolor na biały i pod kątem prostym skręca w lewo do następnej komórki.

2. *Wywołanie programu*
   W celu wywołania programu należy najpierw skorzystać z komendy *make*, która kompiluje kod i tworzy folder na ewentualne pliki wynikowe. Następnie należy wpisać *./edit* i podać argumenty dla programu. Kod nie uruchomi się, jeśli nie podamy*: -m* liczby rzędów ramki, *-n* liczby kolumn, *-i* liczby iteracji mrówki, *-d* kierunku startowego mrówki. Poza tym musimy wprowadzić *-g* procent zapełnienia mapy lub *-p* nazwę pliku z mapą. Program umożliwia nam także zapisanie kolejnych etapów mrówki w plikach wynikowych, których nazwę podajemy w argumencie *-f*. Jeśli nie skorzystamy z tej opcji, wynik zostanie zwrócony na standardowe wyjście. Komenda *make clean* usunie pliki wynikowe oraz skompilowane pliki. W przypadku, gdy chcielibyśmy usunąć jedynie pliki wynikowe możemy skorzystać z komendy *make cleanOut*.

3. *Opis programu*
   Podzieliliśmy program na moduły w celu zwiększenia przejrzystości kodu i uproszczenia poszukiwania ewentualnych błędów. W jednym z nich zdefiniowaliśmy wszystkie zmienne potrzebne do rysowania siatki, mrówki oraz jej kolejnych etapów poruszania się.

```c
#include "characters.h"

wchar_t* LINE_VERTICAL = L"│";
wchar_t* LINE_HORIZONTAL = L"─";
wchar_t* LINE_DOWN_RIGHT = L"┌";
wchar_t* LINE_DOWN_LEFT = L"┐";
wchar_t* LINE_UP_RIGHT = L"└";
wchar_t* LINE_UP_LEFT = L"┘";
wchar_t* SQUARE_WHITE = L" ";
wchar_t* SQUARE_BLACK = L"■";
wchar_t* ARROW_NORTH_WHITE = L"△";
wchar_t* ARROW_NORTH_BLACK = L"▲";
wchar_t* ARROW_EAST_WHITE = L"▷";
wchar_t* ARROW_EAST_BLACK = L"▶";
wchar_t* ARROW_SOUTH_WHITE = L"▽";
wchar_t* ARROW_SOUTH_BLACK = L"▼";
wchar_t* ARROW_WEST_WHITE = L"◁";
wchar_t* ARROW_WEST_BLACK = L"◀";
```

Rys. 1. characters.c odpowiedzialny za rysowanie na ekranie.

Również napisaliśmy moduł odpowiedzialny za obsługę argumentów podawanych przez użytkownika. Funkcja *argError* zawarta w nim sprawdza, czy dana opcja wymaga podania argumentu. Jeżeli użytkownik go nie wprowadzi, zostanie zwrócony odpowiedni komunikat o błędzie i program się zakończy. Zadaniem *intCheck* jest sprawdzenie czy dana wartość przekazana jako argument dla danej opcji jest liczbą. Jeśli nie, również wypisuje komunikat o błędzie i kończy program. Funkcja *argumenty* analizuje argumenty wiersza programu i ustawia zmienne odpowiedzialne za parametry programu. Wykorzystaliśmy tu *getopt,* który pozwala na zautomatyzowane przetwarzanie opcji wiersza poleceń. Dzięki temu program jest łatwy w obsłudze. W celu stworzenia ponumerowanych w odpowiedniej kolejności plików wynikowych wykorzystaliśmy funkcję *filename_Number*. Ostatnie funkcje czyli *fileOut* i *fileIn* służą odpowiednio do zapisu i odczytu mapy do lub z pliku.

```c
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#include <string.h>
#include <unistd.h>
#include <ctype.h>

void argError(char argName, char* value){

    char pValue = *value;


    if (pValue == '-'){
        fprintf(stderr, "Opcja -%c wymaga podania argumentu.\n", argName);
        exit(0);
    }

}

void intCheck(char argName, char* value){

    int i = 0;

    while (value[i] != '\0') {
        if (!isdigit(value[i])){
            fprintf(stderr, "Opcja -%c wymaga podania liczby.\n", argName);
            exit(0);
        }

        i++;
    }

}


void argumenty(int argc, char **argv, int* m, int* n, int* i, char** f, char** d, char** p, int* g){

    // -m (int) wiersze
    // -n (int) kolumny
    // -i (int) iteracje
    // -f (char*) nazwa plikow wyjsciowych (opcjonalnie)
    // -d (char*) kierunek poczatkowy mrowki
    // -p (char*) mapa pokryta czarnymi (opcjonalnie)
    // -g (int) wygenerowanie nowej mapy zapelnionej procentowo czarnymi (opcjonalnie)

    // extern char *optarg;
    // extern int optind, optopt;
    int c;

    int argCheck = 0;
```

```
while ((c = getopt (argc, argv, "m:n:i:f:d:p:g:")) != -1)
    switch (c)
    {
    case 'm':

        argError(c, optarg);
        intCheck(c,optarg);
        *m = atoi(optarg);
        argCheck++;
        break;
    case 'n':
        argError(c, optarg);
        intCheck(c,optarg);
        *n = atoi(optarg);
        argCheck++;
        break;
    case 'i':
        argError(c, optarg);
        intCheck(c,optarg);
        *i = atoi(optarg);
        argCheck++;
        break;
    case 'f':
        argError(c, optarg);
        *f = optarg;

        break;
    case 'd':
        argError(c, optarg);

        if (strcmp(optarg, "w") && strcmp(optarg, "e") && strcmp(optarg, "n") && strcmp(optarg, "s")){
            fprintf (stderr, "Opcja -%c: Nie rozpoznano kierunku. Wybierz kierunek z podanych: {n, s, e, w}.\n", c);
            exit(0);
        }
        *d = optarg;
        argCheck++;
        break;
    case 'p':
        argError(c, optarg);
        *p = optarg;
        optArgCheck++;
        break;
    case 'g':
        argError(c, optarg);
        intCheck(c,optarg);
        *g = atoi(optarg);
        optArgCheck++;
        break;
    case '?':
        if (optopt == 'm' || optopt == 'n' || optopt == 'i' || optopt == 'd' || optopt == 'f' || optopt == 'p' || optopt == 'g'){
            fprintf (stderr, "Opcja -%c wymaga podania argumentu.\n", optopt);

        } else if (isprint (optopt)) {
            fprintf (stderr, "Opcja nieznana `-%c'.\n", optopt);
        }
        exit(0);
    default:
        abort();
    }

if (argCheck < 4){
    fprintf(stderr, "Za malo podanych argumentow. Upewnij sie ze podales argumenty: m, n, i, d\n");
    exit(0);
} else if (argCheck > 5){
```

```
        fprintf(stderr, "Za duzo podanych argumentow.\n");
        exit(0);
    }

    if (optArgCheck < 1){
        fprintf(stderr, "Za malo podanych argumentow: Musisz podac przynajmniej jeden z tych argumentow: g, p\n");
        exit(0);
    } else if (optArgCheck > 1) {
        fprintf(stderr, "Za duzo podanych argumentow: Argumentow g i p nie mozna uzywac na raz\n");
        exit(0);
    }

}


char* filename_Number(char* name, int number) {
    const char* dirPrefix = "output_files/";

    int nameLen = strlen(name);
    int numLen = snprintf(NULL, 0, "_%d", number);

    char* file = (char*)malloc(nameLen + numLen + 2 + strlen(dirPrefix));

    if (file == NULL)
        return NULL;

    strcpy(file, dirPrefix);
    strcat(file, name);
    snprintf(file + nameLen + strlen(dirPrefix), numLen + 2, "_%d", number);




    return file;
}


void fileOut(wchar_t** board, int m, int n, char* name){



    FILE *out = fopen(name, "w");

    for (int i = 0; i < m + 1; i++){
        for (int j = 0; j < n + 1; j++)
            fprintf(out,"%lc", board[i][j]);
        fprintf(out, "\n");
    }

    fclose(out);

}




wchar_t** fileIn(char* name,int m, int n){
    FILE* in = fopen(name, "r");
    wint_t buff;
    wchar_t **board = (wchar_t**)malloc((m + 1) * sizeof(wchar_t*));

     for (int i = 0; i <= m; i++) {
        board[i] = (wchar_t*)malloc((n + 2) * sizeof(wchar_t));
        for (int j = 0; j <= n + 1; j ++){
```

```
                buff = fgetwc(in);
                board[i][j] = buff;
            }

        }


     return board;
}
```

Rys. 2-5. dataMgmt.c odpowiedzialny za obsługę argumentów programu.

Napisaliśmy moduł odpowiedzialny za ruch mrówki i kolorowanie mapy. W kodzie *ant.c* są zdefiniowane funkcje. Pierwsza z nich *dictAnt* przekształca kierunek i kolor mrówki na odpowiadający mu znak Unicode reprezentujący strzałkę o danym kierunku i kolorze. Do czytania koloru komórki służy *Color*. Funkcja *changeColor* zmienia kolor mrówki z białego na czarny lub z czarnego na biały, aby była widoczna na komórce o przeciwnym kolorze. *DirectionNumber* przyjmuje jako argument string reprezentujący kierunek i zwraca odpowiadający mu numer (0 – north, 1 – east, 2 – south, 3 – west). Na końcu znajduje się funkcja *master*, czyli główna funkcja symulująca ruch mrówki. Jako argumenty przyjmuje, wskaźnik do mapy, jej rozmiary, pozycję początkową mrówki, jej początkowy kierunek, liczbę iteracji oraz opcjonalny parametr *f,* czyli nazwę pliku wynikowego. Funkcja iteruje przez określoną liczbę kroków, aktualizując planszę i pozycję mrówki na każdym etapie. W przypadku wyjścia poza granicę ramki, program wypisze błąd i zakończy działanie. Dodatkowo, jeśli zostanie podana nazwa pliku, zapisuje mapę do pliku na każdym kroku iteracji.

```c
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <string.h>

#include "dataMgmt.h"
#include "characters.h"
#include "ant.h"
#include "genBoard.h"

// W - west
// E - east
// N - north
// S - south



wchar_t dictAnt(int direction, char* color) {
    wchar_t result;

    switch (direction) {
        case 3: // WEST
            if (strcmp(color, "black") == 0)
                result = ARROW_WEST_BLACK[0];
            else if (strcmp(color, "white") == 0)
                result = ARROW_WEST_WHITE[0];
            break;

        case 1: // EAST
            if (strcmp(color, "black") == 0)
                result = ARROW_EAST_BLACK[0];
            else if (strcmp(color, "white") == 0)
                result = ARROW_EAST_WHITE[0];
            break;

        case 0: // NORTH
            if (strcmp(color, "black") == 0)
                result = ARROW_NORTH_BLACK[0];
            else if (strcmp(color, "white") == 0)
                result = ARROW_NORTH_WHITE[0];
            break;

        case 2: // SOUTH
            if (strcmp(color, "black") == 0)
                result = ARROW_SOUTH_BLACK[0];
            else if (strcmp(color, "white") == 0)
                result = ARROW_SOUTH_WHITE[0];
            break;

        default:
            printf("\nBlad w kierunkach mrowki ( ant.c/dictAnt() )\n");
            exit(0);
            break;
    }

    return result;
}
```

```c
char* Color(wchar_t square){
    if (square == SQUARE_BLACK[0])
        return "black";
    else
        return "white";
}

wchar_t changeColor(char* color){
    if (strcmp(color, "white") == 0)
        return SQUARE_BLACK[0];
    else
        return SQUARE_WHITE[0];

}


int DirectionNumber (char* direction){
    if (strcmp(direction, "n") == 0)
        return 0;
    else if (strcmp(direction, "e") == 0)
        return 1;
    else if (strcmp(direction, "s") == 0)
        return 2;
    else if (strcmp(direction, "w") == 0)
        return 3;
}

void master(wchar_t** board,int m, int n, int x, int y, int iteration, char* startDirection, char* f){

    int k = DirectionNumber(startDirection);

    char* color;




    if (board[x][y] == ARROW_EAST_BLACK[0] || board[x][y] == ARROW_WEST_BLACK[0] || board[x][y] == ARROW_NORTH_BLACK[0] || board[x][y] == ARROW_SOUTH_BLACK[0])
        color = "black";
    else
        color = "white";



    for (int i = 0; i < iteration; i ++){

        if (f != NULL){
            char* name = filename_Number(f, i);
            fileOut(board, m, n, name);
        }
        else
            boardOut(board,m,n);

        if (strcmp(color, "white") == 0){
            k += 1;
            if (k > 3)
                k = 0;
        }
        else if (strcmp(color, "black") == 0){
            k -= 1;
            if (k < 0)
                k = 3;
        }
```

```c
        switch (k){
            case 0: // NORTH
                board[x][y] = changeColor(color);
                x -= 1;
                break;
            case 1: // EAST
                board[x][y] = changeColor(color);
                y += 1;
                break;
            case 2: // SOUTH
                board[x][y] = changeColor(color);
                x += 1;
                break;
            case 3: // WEST
                board[x][y] = changeColor(color);
                y -= 1;
                break;

        }

        if (x <= 0 || x >= m || y <= 0 || y >= n){
            printf("Mrowka wyszla poza granice planszy.\nIlosc przejsc: %d\n", i);
            exit(0);
        }
        color = Color(board[x][y]);
        board[x][y] = dictAnt(k, Color(board[x][y]));

    }



}
```

Rys. 6-8. ant.c odpowiedzialny za poruszanie się mrówki po mapie.

Ostatnim modułem był kod odpowiedzialny za generowanie mapy. Składa się on między innymi z funkcji *randrange* do losowania liczby całkowitej z podanego zakresu. Funkcja *boardOut* przyjmuje dwuwymiarową tablicę oraz jej rozmiary i wypisuje jej zawartość na konsolę. Do odnalezienia wskazanego elementu w

określonym zakresie na mapie stworzyliśmy funkcję *binarySearch,* którą wykorzystaliśmy w *korekta*, aby odnajdywać białe pola i zamieniać je na czarne do momentu aż zostanie osiągnięta żądana liczba czarnych pól. Do zwalniania alokowanej pamięci dla dwuwymiarowej tablicy służy *freeBoard*, a do znajdywania pozycji mrówki na mapie – *arrowSearch*. Funkcja *genMap* to główna funkcja generująca mapę o danych wymiarach, z określonym procentem czarnych pól oraz mrówką umieszczoną na pozycji (*antX, antY*) i zwracającą się w określonym kierunku. Na początku funkcja inicjalizuje mapę z granicami i wypełnia ją losowo, przy użyciu *randrange*, białymi i czarnymi polami zgodnie z podanym procentem. Na koniec kodu, zostaje wywołana funkcja *korekta* w celu dostosowania planszy do żądanej liczby czarnych pól oraz zostaje ustawiona mrówka na mapie.

```c
#include "genBoard.h"
#include "ant.h"

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "characters.h"



int randrange(int a, int b) {

    int randomNum = rand() % b + a;

    return randomNum;
}
void boardOut(wchar_t **board, int m, int n){

    for (int i = 0; i < m + 1; i++){
        for (int j = 0; j < n + 1; j++)
            printf("%lc", board[i][j]);

        printf("\n");
    }


}

int binarySearch(wchar_t *board, int low, int high, wchar_t target) {
    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (board[mid] == target)
            return mid;

        if (board[mid] < target)
            low = mid + 1;

        else
            high = mid - 1;
    }

    return -1;
}

void korekta(wchar_t ** board, int blackCounter, int m, int n){

    int i = 0;
    int result;
    int check = 0;
    while (blackCounter > 0){

        result = binarySearch(board[i], 1, n - 1, SQUARE_WHITE[0]);
        if (result != -1){
            board[i][result] = SQUARE_BLACK[0];
            blackCounter -= 1;

        } else
            i += 1;

        }



}
```

```c
void freeBoard(wchar_t** board, int rows) {
    for (int i = 0; i < rows; i++) {
        free(board[i]);
    }
    free(board);
}



void arrowSearch(wchar_t** board, int m, int n, int* x, int* y){

   for (int i = 1; i < m; i ++){
        for (int j = 1; j < n; j ++)
            if (board[i][j] != SQUARE_BLACK[0] && board[i][j] != SQUARE_WHITE[0]){
                *x = i;
                *y = j;
                break;
            }
   }
}




wchar_t** genMap(int m, int n, int percent, int antX, int antY, char* direction) {
    srand(time(NULL));

    int dir = DirectionNumber(direction);

    m += 1;
    n += 1;

    float blackChance = percent;
    int blackCount = round(((m - 2) * (n - 2)) * (blackChance/100));


    wchar_t **board = (wchar_t**)malloc(m * sizeof(wchar_t*));

    for (int i = 0; i <= m; i++) {
        board[i] = (wchar_t*)malloc(n * sizeof(wchar_t));

        for (int j = 0; j < n; j++) {
            if (i == 0){
                if (j == 0)
                    board[i][j] = LINE_DOWN_RIGHT[0];
                else if (j == n - 1)
                    board[i][j] = LINE_DOWN_LEFT[0];
                else
                    board[i][j] = LINE_HORIZONTAL[0];
            }
            else if (i == m - 1){
                if (j == 0)
                    board[i][j] = LINE_UP_RIGHT[0];
                else if (j == n - 1)
                    board[i][j] = LINE_UP_LEFT[0];
                else
                    board[i][j] = LINE_HORIZONTAL[0];

            }

            else if (j == 0 || j == n - 1){
                board[i][j] = LINE_VERTICAL[0];
            }

            else{
                if (randrange(1, 100) <= blackChance && blackCount > 0){
                    board[i][j] = SQUARE_BLACK[0];
                    blackCount -= 1;

                }

                else{
                    board[i][j] = SQUARE_WHITE[0];
                }

            }

        }


    }

    korekta(board,blackCount,m, n);


    board[antX][antY]  = dictAnt(dir, Color(board[antX][antY]));


    return board;


}
```

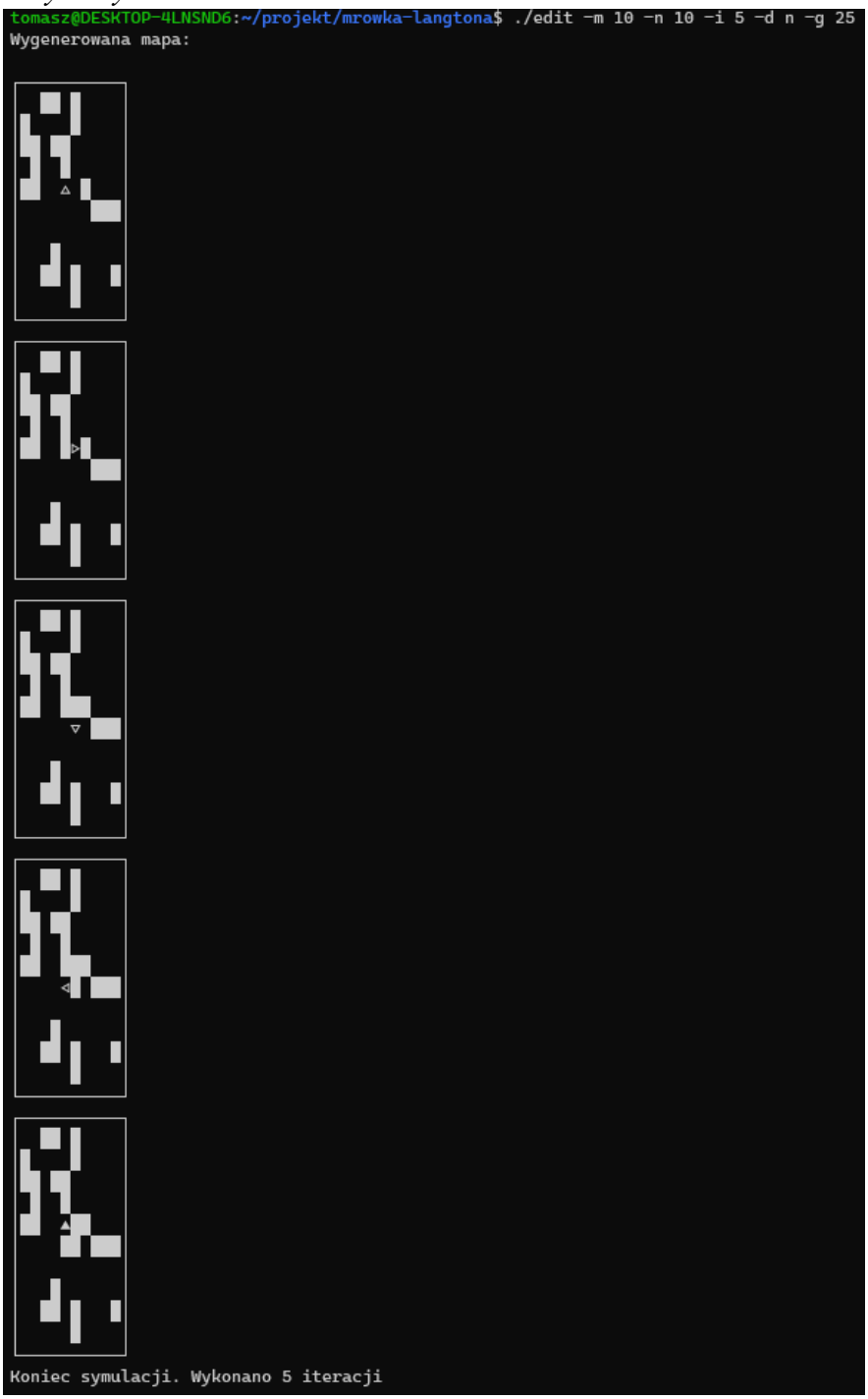
Rys. 9,10. genBoard.c odpowiedzialne za stworzenie mapy do symulacji ruchu mrówki.

W pliku *main.c* zostają wywołane wszystkie potrzebne funkcje do działania programu. Na początku zostaje ustawiona lokalizacja na „C.UTF-8”, aby działały polskie znaki i znaki z *characters.*c. Potem odczytywane są argumenty podane przez użytkownika. W przypadku generowania mapy, pozycja mrówki jest ustawiana na środku mapy. Jeżeli program wczytuje mapę to odszukujemy komórkę, w której znajduje się mrówka. Następnie mapa jest wyświetlana na ekranie i program przechodzi do przeprowadzenia symulacji mrówki. Na koniec użytkownik zostaje poinformowany o zakończonej symulacji i liczbie wykonanych iteracji.

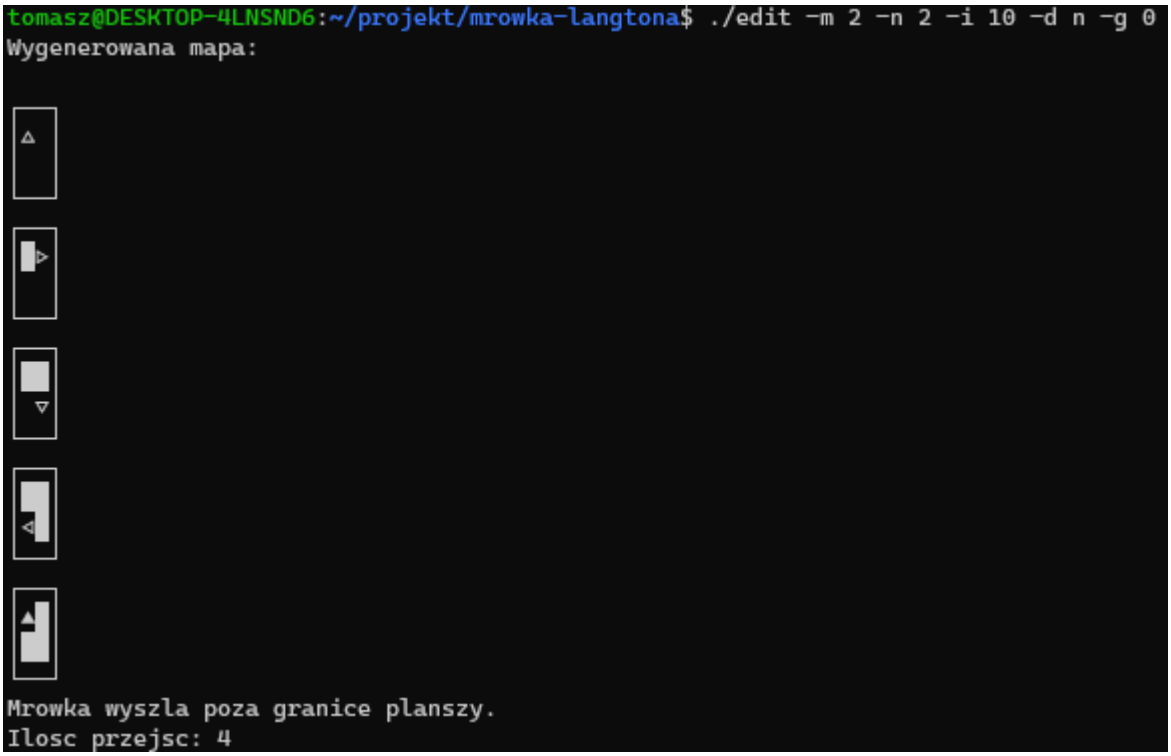
```c
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#include <string.h>
#include <math.h>
#include "characters.h"

#include "ant.h"
#include "genBoard.h"
#include "dataMgmt.h"


int main(int argc, char **argv) {
    setlocale(LC_ALL, "C.UTF-8");


    int m, n, i, g = 0;
    char *f = NULL, *d, *in, *p = NULL;

    argumenty(argc, argv, &m, &n, &i, &f, &d, &p, &g);


    int x = round((m)/2);
    int y = round((n)/2);


    wchar_t** mapa;
    m += 1;
    n += 1;

    if (p == NULL){
        // procent i kierunek

        printf("Wygenerowana mapa:\n\n");
        mapa = genMap(m, n, g, x, y, d);


        if (f != NULL){
            boardOut(mapa, m, n);
            fileOut(mapa, m, n, f);
        }
        master(mapa, m, n, x, y, i, d, f);


    } else {
    // preset
        printf("Wczytana mapa:\n\n");

        mapa = fileIn(p, m, n);
        if (f != NULL)
            boardOut(mapa, m, n);
        arrowSearch(mapa, m, n, &x, &y);
        master(mapa, m, n, x, y, i, d, f);
    }



    printf("Koniec symulacji. Wykonano %d iteracji\n", i);


    return 0;
    }
```

Rys. 11. Kod main.c odpowiedzialny za działanie całego programu.

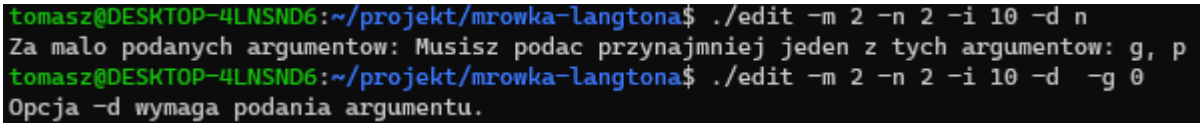
4. *Przykłady*



Rys. 12. Przykład działania programu dla mapy 10x10, 5 iteracji, mrówki skierowanej w górę i wygenerowanej mapy w 25% w czarnych polach.



Rys. 13. Przykład działania programu, gdy mrówka wyjdzie za mapę.



Rys. 13. Przykład działania programu przy błędnym wpisaniu argumentów.

5. *Wnioski*

   Podczas projektu zauważyliśmy, że skuteczny podział obowiązków może zaoszczędzić dużo czasu, a współpraca może się okazać przyjemna i efektowna. Poza tym utrwaliliśmy programowanie w języku C oraz komendy git'a. Ponadto doświadczyliśmy tego, jak komunikacja w zespole jest ważna podczas projektu. Uważamy, że projekt mrówki Langtona wiele nas nauczył i możemy go uznać za zakończony sukcesem.