Łukasz Jarzęcki Paweł Skierkowski

MRÓWKA LANGTONA

Zasady działania algorytmu:

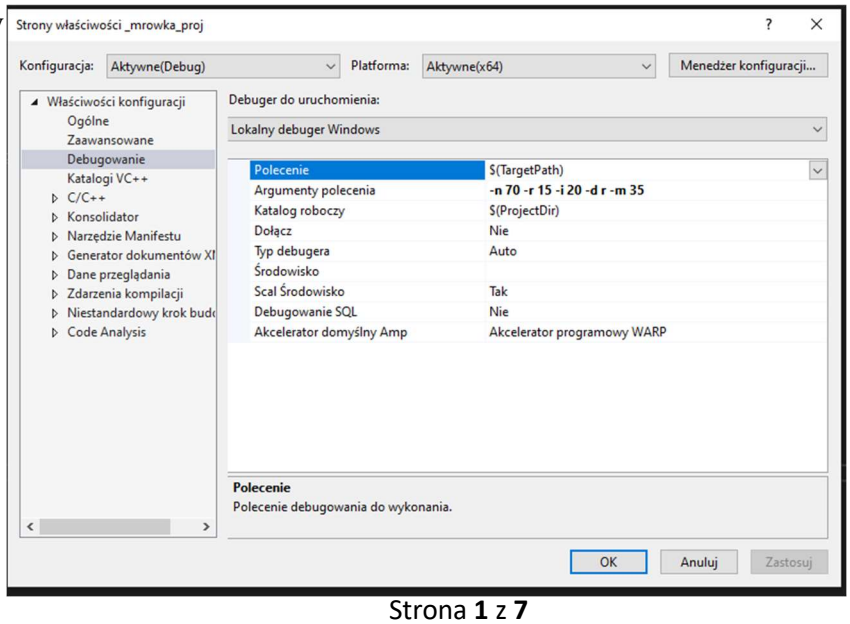
Mrówka przemieszcza się po dwuwymiarowej siatce. Może poruszać się w jednym z 4-ech kierunków (góra, dół, lewo, prawo), zgodnie z następującymi zasadami:

- Jeżeli mrówka znajduje się na białej komórce wykonuje: obrót o 90 stopni w prawo (zgodnie z ruchem wskazówek zegara), zmienia kolor komórki na przeciwny oraz przesuwa się o jedną komórkę do przodu.
- Jeżeli mrówka znajduje się na czarnej komórce wykonuje: obrót o 90 stopni w lewo (przeciwnie do ruchu wskazówek zegara), zmienia kolor komórki na przeciwny oraz przesuwa się o jedną komórkę do przodu
- Mrówka nie może wyjść poza granicę planszy, dlatego w przypadku, gdy mrówka miałaby przemieścić się poza granicę algorytm kończy swoje działanie

Wywoływanie programu:

Aby wywołać program z powłoki konsoli należy skompilować program a następnie wywołać go z odpowiednimi parametrami podanymi poniżej.

Innym sposobem na wywołanie programu jest stworzenie projektu w visual studio, dodanie plików źródłowych, a następnie wejście we właściwości projektu i podanie parametrów w oknie ”argumenty



Program jest przystosowany do uruchomienia na systemie Windows.

Parametry możliwe do podania podczas uruchamiania to:

-m -liczba rzędów planszy /domyślnie 50

-n -liczba kolumn planszy /domyślnie 50

-i -ilość iteracji /domyślnie 20

-f -przedrostek pliku wyjściowego /domyślnie program wypisuje na stdout /nie podajemy rozszerzenia a jedynie nazwę – domyślnie tworzy pliki .txt

-l nazwa pliku wejściowego /jeżeli chcemy aby plansza została wygenerowana automatycznie pomijamy parametr

-r <0,100>- procentowa ilość czarnych pól wybranych losowo /domyślnie 0 /pomijane przy wczytywaniu planszy

-d <u/d/l/r> - początkowy kierunek mrówki /domyślnie mrówka skierowana jest w górę /pomijane przy wczytywaniu planszy

Podział programu na moduły:

Program podzielony jest na moduły:

- board.c board.h //odpowiedzialny za inicjację, generowanie lub wczytywanie planszy
- fhandling.c fhandling.h //odpowiedzialne za obsługę plików
- macro.h //zawiera specjalne znaki oraz załącza biblioteki
- main.c //główne ciało programu
- move.c move.h //odpowiedzialne za przechodzenie mrówki po planszy
- getopt.c getopt.h //implementacja getopta dla visual studio

Przykładowe funkcje wraz z opisem:

```c
char* fileName(char* name, int it)
{

    if (strcmp(name, "stdout") == 0)
        return "stdout";

    int itlen = (int)(log10((double)(it))) + 1;

    char* fname = malloc(sizeof(name) + 1 + MAX_IT_LEN);
    if (fname == NULL)
    {
        fprintf(stderr, "Nie moge stworzyc nazwy pliku");
        return 1;
    }

    sprintf(fname, "%s_%d.txt", name, it);

    return fname;
}
```

Jest to funkcja służąca do zwracania nazwy pliku po podania przedrostka i iteracji.

```c
void printBoard(wchar_t** board, int r, int c, char *fname)
{


    FILE* out;
    if (strcmp(fname, "stdout") == 0)
    {
        out = stdout;
        system("cls");
    }

    else
    {
        out = fopen(fname, "w");
    }

    if (out == NULL)
    {
        fprintf(stderr, "Nie moge stworzyc pliku");
        return 1;
    }

    r += 2;
    c += 2;

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

Funkcja printBoard służy do wypisywania elementów tablicy (w zależności od podanych argumentów robi to w konsoli rób zapisuje do pliku)

```c
wchar_t** boardInit(int r, int c)
{
    wchar_t** board = malloc(r * sizeof(wchar_t*));
    if (board == NULL)
    {
        fprintf(stderr, "Nie moge stworzyc planszy");
        return 1;
    }
    for (int i = 0; i < r; i++)
    {
        board[i] = malloc(c * sizeof(wchar_t));
        if (board[i] == NULL)
        {
            fprintf(stderr, "Nie moge stworzyc planszy");
            return 1;
        }
    }
    return board;
}
```

Funkcja boardInit służy do inicjacji planszy. Wykorzystuje parametry r i c będące wymiarami planszy, a następnie przydziela tablicy pamięć.

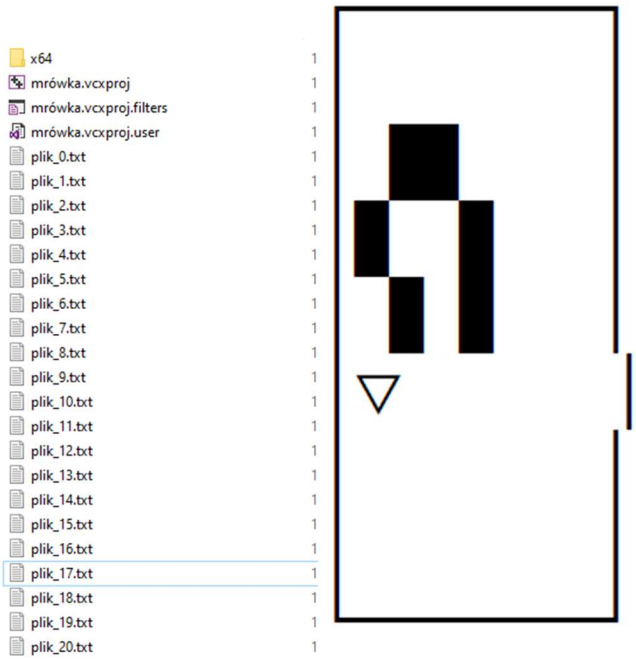
```c
#include "move.h"

void przejscie(wchar_t **board,int r,int c)
{
    for (int i = 1; i < r-1; i++)
    {
        for (int j = 1; j < c-1; j++)
        {
            if (board[i][j] == ARROW_NORTH_WHITE)
            {
                if (board[i][j + 1] == VERTICAL_L || board[i][j + 1] == HORIZONTAL_L)
                    exit(EXIT_FAILURE);
                board[i][j] = BLACK_SQUARE;
                if (board[i][j+1] ==WHITE_SQUARE)
                    board[i][j+1] = ARROW_EAST_WHITE;
                else
                    board[i][j+1] = ARROW_EAST_BLACK;
                return;
            }
```

Powyżej widnieje fragment funkcji przejście odpowiadającej za główne działanie algorytmu. Przyjmuje ona planszę oraz jej wymiary. Następnie za pomocą pętli przechodzi po kolejnych wartościach tablicy dwuwymiarowej (zaczynając od drugiego wiersza i drugiej kolumny ponieważ zajęte one są przez obramowanie). Funkcja zgodnie z założeniami algorytmu porównuje pola planszy. Po pierwsza sprawdza czy kolejne pole nie jest granicą i kończy działanie programu, kiedy napotka krawędź (exit(EXIT_FAILURE)). Zmienia kolor pola na którym się znajduje na przeciwny, obraca się w odpowiednim kierunku i przesuwa się na kolejne pole.

Przykładowe działanie programu dla różnych ustawień:

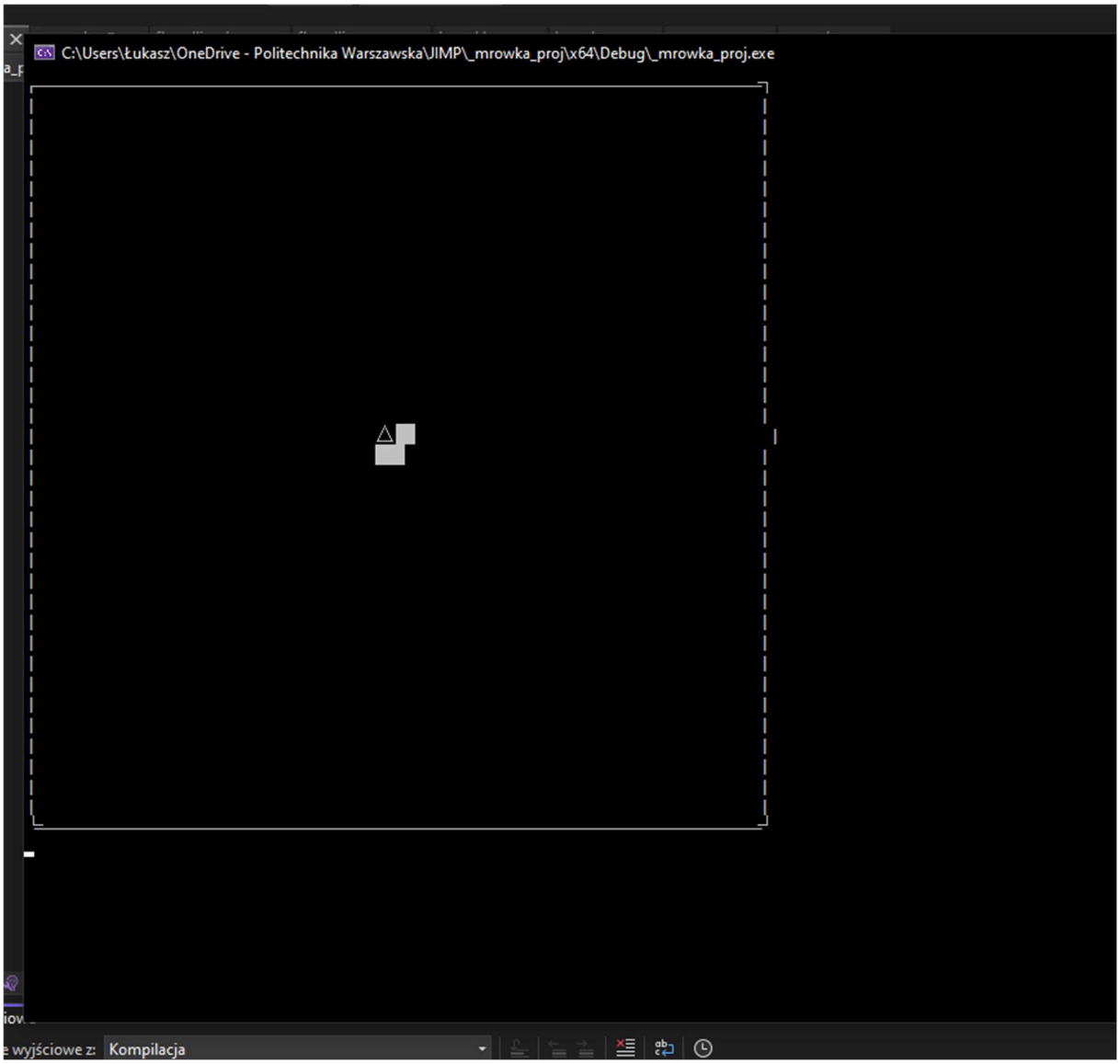
Argumenty polecenia	**-n 10 -r 15 -i 21 -d u -m 8 -f plik**

Dla wpisanego przedrostka pliku program utworzył 20 plików +1 z początkowym położeniem (plik_0.txt) (z powodu wejścia na granicę planszy nie ma pliku plik_21.txt)



W przypadku pominięcia przedrostka w argumentach wywołania program wypisuje kolejne przejścia w konsoli.

Polecenie	$(TargetPath)
Argumenty polecenia	**-n 10 -r 15 -i 21 -d u -m 8**
Katalog roboczy	$(ProjectDir)



Po każdym przejściu konsola jest czyszczona

Wnioski:

Praca w grupie nad projektem ma wiele zalet. Podział obowiązków, przystosowywanie się do przyszłej pracy, w której będziemy musieli współpracować z innymi programistami. Dużym ułatwieniem jest korzystanie z repozytorium na github. W bardzo prosty sposób mogliśmy zamieszczać gotowe rozwiązania i konsultować swoje bieżące postępy w pracy nad modułami.

Algorytm pomimo braku dużej złożoności był ciekawy. Nieraz napotkaliśmy problemy, jednak po długich staraniach i wspólnych wysiłkach udało się je rozwiązać.