Dokumentacja poprojektowa

Opis funkcjonalności
Warunki poprawnego działania programu :
- uruchamianie w dowolnej dystrybucji systemu Linux
- podanie poprawnych danych wejściowych do programu

Uruchomianie i działanie programu:

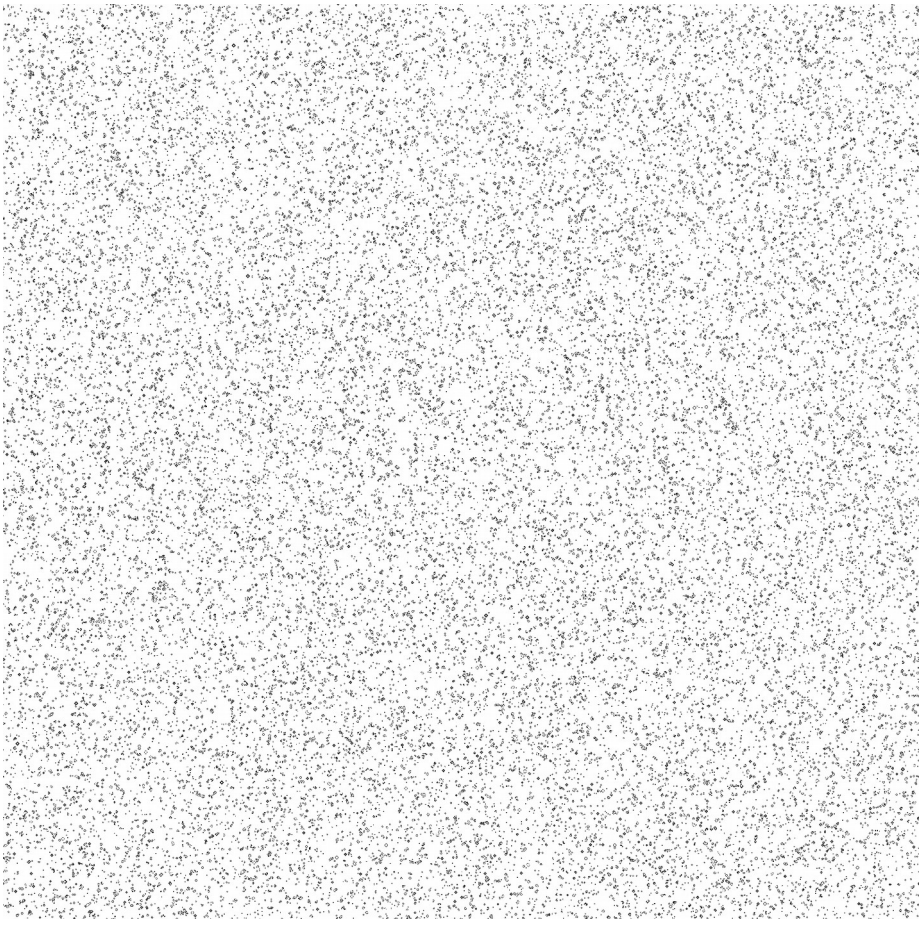
Uruchamiamy program wpisując w konsoli ‘make game’. Wówczas program zostanie uruchomiony z określonymi argumentami tj.

./game 25 30 points 15

gdzie poszczególne argumenty oznaczają :
25 – jest szerokością planszy
30 – jest jej wysokością
points – plik z współrzędnymi ‘żywych’ komórek
15 – liczba iteracji gry do zasymulowania

Program z otrzymanych argumentów wywołania zwróci n plików *.png obrazujących poszczególne iteracje gry.

Przykładowy obraz *.png (dla ilości punktów sięgającej rzędu 0.5 mln) :



Scenariusz działania programu :

1. Program pobiera dane z pliku podanego jako argumenty wywołania i przechowuje je we wcześniej zainicjowanych strukturach danych, w przypadkach:
    a) braku podania danych :
        poinformuje użytkownika o braku i poprosi o podanie danych
    b) błędnych danych :
        zwróci kod błędu i zakończy działanie.

2. Program na podstawie danych wejściowych wyznacza stany gry dla każdej iteracji, która jest wymagana przez użytkownika. W trakcie symulacji wykonywany jest również proces tworzenia plików graficznych reprezentujących stan danej iteracji. Każdy plik jest zapisywany do katalogu obok pliku Makefile.

Format Danych i Struktury plików

katalog src:

- pliki źródłowe :
    - filemanagement.c
    - grid.c
    - main.c
    - neighbours.c
    - pointlist.c
    - simulate.c
- pliki nagłówkowe :
    - filemanagement.h
    - grid.h
    - neighbours.h
    - pointlist.h
    - simulate.h

katalog tests:

- katalog inputFiles:
    - 1
    - 2
    - 3
    - 4
    - 5
    - 6
    - points

Przechowywanie danych w programie i poza programem

Współrzędne dla ‘żywych’ komórek do programu znajdują się w pliku ‘points’
(nazwa pliku z punktami może być inna – w tym przypadku zaleca się użyć komendy :
./bin/game szerokosc_planszy wysokosc_planszy nazwa_pliku_z_punktami ilosc_iteracji).

Przykładowa zawartość powyższego pliku :

Domyślnie wartości dla punktów znajdują się w pliku ‘points’
Przykładowa zawartość pliku ‘points’ :

9 7
9 8
9 9
8 9
7 8
100 5
18 10
1 -1
2 5
8 16
9 18
10 20
1 4
1 9
3 7
18 9
16 4
16 8
15 7
12 13

Struktura pliku ‘points’ :

pierwsza współrzędna komórki, druga współrzędna kom.

Scenariusz komunikacji użytkownika z programem

Opis ogólny :

1) program wczytuje dane z pliku podanego jako jednego z argumentów wywołania i na ich podstawie zmienia odpowiednie pola w zainicjowanych wcześniej strukturach
2) symuluje przebieg gry i tworzy graficzne reprezentacje poszczególnych iteracji
3) zapisuje utworzone pliki *.png do katalogu zawierającego plik ‘Makefile’

Opis szczegółowy :

1) Użytkownik podaje argumenty wywołania :
   - w przypadku braku podania któregoś argumentu, program wyświetli komunikat o takowej konieczności i poprosi o wczytanie brakującego argumentu
   - w przypadku błędnych danych wyświetlony zostanie komunikat o błędzie argumentów i program zakończy działanie
2) Program na podstawie 2 pierwszych argumentów inicjuje struktury, po czym korzystając z pozostałych symuluje grę.
3) Program tworzy graficzne reprezentacje n początkowych iteracji i zapisuje je w katalogu obok pliku ‘Makefile’. Liczba n jest ostatnim parametrem podanym jako argument wywołania.

Zmienne i struktury programu

- Grid – zmienna typu ‘struktura’, będąca odpowiednikiem planszy. Przechowuje dane z pliku podane przez użytkownika jako argumenty wywołania.
Przechowuje wartości typu ‘unsigned integer’ oraz ‘char**’:
    unsigned int x – szerokość planszy
    unsigned int y – wysokość planszy
    char** v – macierz trzymająca stany poszczególnych komórek (plansza)

- PointList – zmienna typu ‘struktura’, przechowująca współrzędne ‘żywych’ komórek znajdujących się na planszy.
Przechowuje wartości typu ‘integer’ oraz ‘struct PointList*’:
    int x,y – odpowiednio pierwsza i druga współrzędna komórki
    struct PointList* next – wskaźnik na następny element listy ‘żywych’ komórek.

- Simulator – zmienna typu ‘struktura’, służąca do utworzenia następnego stanu planszy. Przechowuje wartości typu ‘Grid_t’ (wskaźnik na strukturę Grid) oraz ‘Plist_l’ (wskaźnik na strukturę PointList)

    Grid_t grid – plansza z jej aktualnym stanem
    Grid_t nextgrid – pusta plansza, na której nanoszone są ‘żywe’ komórki (przechowuje nowy stan planszy)
    Plist_l pointlist – dynamiczna lista przechowująca współrzędne ‘żywych’ komórek

Funkcje programu

1) initGrid
Prototyp : Grid_t initGrid(int x, int y);
Lokalizacja : grid.h
Opis : Funkcja inicjuje planszę o zadanych rozmiarach po czym zwraca wskaźnik na utworzoną strukturę .

2) destroyGrid
Prototyp : void destroyGrid(Grid_t g);
Lokalizacja : grid.h
Opis : Funkcja zwalnia pamięć wcześniej zaalokowaną na potrzeby utworzenia planszy w programie.

3) clearGrid
Prototyp : void clearGrid(Grid_t g);
Lokalizacja : grid.h
Opis : Funkcja wypełnia zadaną planszę zerami - ‘uśmierca’ wszystkie komórki, przygotowując planszę do zaktualizowania jej stanu.

4) printGrid
Prototyp : void printGrid(Grid_t g);
Lokalizacja : grid.h
Opis : Funkcja wypisuje stan planszy na ekran konsoli.

5) readFile
Prototyp : void readFile(FILE * in, Simulator_t sim);
Lokalizacja : filemanagement.h
Opis : Funkcja wczytuje dane z plików do struktur wewnętrznych.

6) createPng
Prototyp : void createPng(char* filename, Grid_t grid);
Lokalizacja : filemanagement.h
Opis : Funkcja tworzy plik *.png obrazujący stan planszy w danej iteracji.

7) countNeighbours
Prototyp : int countNeighbours(int x, int y, Grid_t grid);
Lokalizacja : neighbours.h
Opis : Funkcja liczy ilość ‘żywych’ komórek sąsiadujących z komórką o współrzędnych x,y otrzymanych jako parametry.

8) initPlist
Prototyp : Plist_l initPlist(int x, int y);
Lokalizacja : pointlist.h
Opis : Funkcja inicjuje listę ‘żywych’ komórek potrzebną do aktualizacji
stanu planszy.

9) addToPlist
Prototyp: Plist_l addToPlist(Plist_l list, int x, int y);
Lokalizacja : pointlist.h
Opis : Funkcja dodaje komórkę o współrzędnych x,y do list ‘żywych’ komórek.

10) destroyPlist
Prototyp: void destroyPlist(Plist_l list);
Lokalizacja : pointlist.h
Opis : Funkcja czyści listę ‘żywych’ komórek, po czym dealokuje używaną pamięć.

11) initSimulator
Prototyp : Simulator_t initSimulator(int x, int y);
Lokalizacja : simulator.h
Opis : Funkcja inicjuje simulator wraz z inicjacją jego pól
– dwóch plansz oraz listy komórek.

12) prepareSimulation
Prototyp : Simulator_t prepareSimulation(FILE* file);
Lokalizacja : simulator.h
Opis : Funkcja przygotowuje simulator do przeprowadzenia symulacji gry.

13) simulate
Prototyp : int simulate(FILE * file, int x, int y, int iterations);
Lokalizacja : simulator.h
Opis : Funkcja otrzymuje plik z współrzędnymi komórek, wymiary planszy oraz
liczbę iteracji do wykonania. Wewnątrz wywoływane są funkcje tworzące
simulator, na którego bazie przeprowadza się właściwą symulację.
W jej trakcie tworzy się pliki *.png odpowiednich iteracji gry,
wywołując funkcję createPng().

14) calculateFrame
Prototyp : Simulator_t calculateFrame(Simulator_t sim);
Lokalizacja : simulator.h
Opis : Funkcja mając aktualny stan planszy, listę ‘żywych’ komórek tworzy następny
stan planszy.

15) destroySimulator
Prototyp : void destroySimulator(Simulator_t simulator);
Lokalizacja : simulator.h
Opis : Funkcja niszczy simulator przez dealokację pamięci.

Wnioski
Założenia projektu zostały zrealizowane. Program wykonuje poprawnie wszystkie zaimplementowane funkcje. Analizując poszczególne pliki *.png trzeba zauważyć że założenia Conway'a dotyczące 'gry w życie' zostały spełnione.