

# Automat komórkowy CellSim

## Sprawozdanie

### 1. Wstęp merytoryczny i koncepcja programu

Celem programu jest symulacja automatu komórkowego wzorowanego na Game of Life Johna Conwaya. Taki typ automatu charakteryzuje się dwuwymiarową, nieskończoną siatką, na której rozmieszczone są komórki mogące znajdować się w jednym z dwóch stanów. Komórkę martwą oznaczać będziemy kolorem białym, a żywą kolorem czarnym. Każda następna generacja ustalana jest na podstawie obecnego stanu komórek według sąsiedztwa Moore’a (bada się osiem najbliższych sąsiadów każdej komórki), lub sąsiedztwa von Neumanna (bada się cztery przylegające komórki). Dla uproszczenia program będzie symulował ograniczoną siatkę. Komórka pozostaje żywa, jeżeli ma dwóch, lub trzech sąsiadów, w przeciwnym wypadku umiera. Komórka martwa staje się żywa, jeżeli ma dokładnie trzech sąsiadów. Użytkownik ma możliwość ustalenia sąsiedztwa, wymiarów siatki, zachowania przy krawędziach siatki, ilości symulowanych generacji, pliku wejściowego z początkowym stanem automatu, oraz pliku wyjścia, w którym w formacie png zapisywany będzie stan automatu w ostatniej symulowanej generacji, lub ewentualnie istnieje możliwość wprowadzenia interwału, w którym generowane mają być pliki zawierające aktualny stan automatu.

### 2. Uruchamianie i dane wejściowe


```
./cellsim 1 2
```

1. nazwa pliku z danymi konfiguracyjnymi
2. nazwa pliku ze stanem początkowym

#### 2.1. Dane konfiguracyjne

Plik posiada “komendy” podawane w kolejnych liniach postaci:

`dana_konfigurowana = wartość`

 cholewp1@jim

```
nbrhd = Moore  
bound = 1  
save = 10  
intrvl = 5  
m = 100  
n = 100
```

**Możliwe dane do skonfigurowania:**

**Typ sąsiedztwa:**

nbrhd = { Moore , Neumann , 4 , 8 }

Moore , 8           -> sąsiedztwo Moore'a

Neumann , 4       -> sąsiedztwo Neumanna

**Warunki graniczne:**

bound = { 0 , -1 }

0                   -> komórki poza zakresem są traktowane jak martwe

-1                   -> plansza zapętla się

**Save:**

save = { /liczba naturalna/ }

ilość zapisów

**Interwał zapisu:**

intrvl = { /liczba naturalna/ }

częstotliwość zapisu

**Ilość kolumn:**

m = { /liczba naturalna/ }

### Ilość wierszy:

$n = \{ \text{/liczba naturalna/} \}$

### Szablon nazw plików wyjściowych:

$out = \{ \text{/nazwa/} \}$

szablon będzie postaci:

$\text{/numer\_zapisu/\_nazwa/}$

zaleca się kończenie nazw “.png”

### Symulacja na żywo:

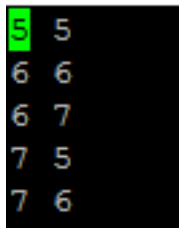
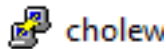
$live = \{ \text{yes , /nothing/} \}$

yes -> będzie symulacja na żywo

/nothing/ -> nie będzie symulacji na żywo

## 2.2. Plik planszy

Zawiera w kolejnych liniach współrzędne kolejnych komórek żywych.



## 3. Moduły programu

### Sterowanie

#### main.c

1. `int main(int argc, char **argv);`

## Opis struktur głównych

### struct.h

```
1. typedef struct {
    int nbrhd;          /* typ sąsiedztwa (4,8) */
    int bound;
    /* typ warunków brzegowych (0,-1) */
    int save;           /* częstotliwość zapisu */
    char *out           /* nazwa pliku wyjścia */
    int intrvl;         /* interwał czasowy zapisu */
    char live;          /* czy będzie symulacja na żywo */
} cfg_t;

2. typedef struct {
    char **board;
    /*wiersze tablicy to wiersze komórek
    kolumny tablicy zawierają indeksy
    kolumn komórek wierszy
    zerowy wiersz zawiera
    tablica jest mxn
    */
    char **old_boafd
    /* jak wyżej tylko dla stanu wcześniej */
    int m;              /* ilość wierszy planszy */
    int n;              /* ilość kolumn planszy */
    cfg_t cfg;
} *map_t;
```

## Alokator pamięci i wczytanie danych wejściowych

### alloc.c (.h)

```
1. map_t initalloc (map_t map );
```

- alokacja struktury
  - nadanie wartości domyślnych
2. `map_t boardalloc( map_t map );`
    - alokacja tablic
  3. `map_t add_cfg( map_t map , FILE *in );`1
    - czytanie danych konfiguracyjnych z pliku
  4. `map_t add_map( map_t map , FILE *in );`
    - wczytanie stanu początkowego
  5. `void freealloc( map_t map );`
    - zwolnienie miejsca z pamięci

## Generator główny

### gen.c (.h)

1. `map_t iter_gen(map_t map);`
  - wywołuje generowanie pokoleń
  - iteruje po generacjach
  - wywołuje zapisania stanu
2. `map_t next_gen( map_t map);`
  - zwraca następną generację

## Moduły generacyjne

### nbr.c (.h)

1. `int num_of_neighbours( map_t map, int x, int y );`
  - sprawdza rodzaj sąsiedztwa
  - zwraca ilość wartości odpowiedniej funkcji liczącej sąsiadów
2. `int neumann_neighbourhood( map_t map, int x, int y );`
  - zwraca ilość sąsiadów według sąsiedztwa von Neumanna
3. `int moore_neighbourhood( map_t map, int x, int y );`
  - zwraca ilość sąsiadów według sąsiedztwa Moore'a

### **life.c (.h)**

1. int cells\_next\_state(map\_t map, int x, int y );
  - zwraca następny stan komórki o współrzędnych (x, y) - 0-bez zmian, -1-umiera, 1-staje się żywa

### **bound.c (.h)**

1. int check\_state( map\_t map , int x ,int y);
  - zwraca stan komórki o podanych współrzędnych
  - wykorzystuje do tego poniższe funkcje
  - sprawdza graniczność planszy
2. int dead\_bound( map\_t map, int x , int y);
  - zwraca stan są komórki o podanych współrzędnych, uwzględniając komórki poza zakresem, umartwiając potencjalne
3. int live\_bound( map\_t int x, int y);
  - zwraca stan są komórki o podanych współrzędnych, uwzględniając komórki poza zakresem, wskrzeszając potencjalne

## **Moduły wyjściowe**

### **live.c (.h)**

1. void print\_board( map\_t map );
  - czyści konsolę
  - wykonuje krótkie czekanie
  - wyświetla aktualny stan planszy
  - granice planszy są symbolizowane przez symbole “|” i “-”
  - komórki żywe są symbolizowane przez “#”
  - komórki martwe nie wyświetlają się

### **save.c (.h)**

1. int scale( int m, int n);
  - zwraca skalę (1:1, 3:1, lub 5:1) w zależności od rozmiaru planszy (dla lepszej widoczności dla małych i średnich plansz)
2. int save( map\_t map );
  - wybór nazwy pliku i miejsca zapisu w zależności od ustawień

- przekazanie potrzebnych danych do odpowiednio skalującej funkcji
  - konwersja pliku tymczasowego pbm do pliku png (linuxowe narzędzie convert)
  - usunięcie pliku tymczasowego
3. void save\_\_scalex1( map\_\_t map );
    - zapis tablicy \*\*board do graficznego pliku tymczasowego pbm w skali 1:1
  4. void save\_\_scalex3( map\_\_t map );
    - zapis tablicy \*\*board do graficznego pliku tymczasowego pbm w skali 3:1
  5. void save\_\_scalex5( map\_\_t map );
    - zapis tablicy \*\*board do graficznego pliku tymczasowego pbm w skali 5:1

## 4.Struktura katalogów

### Katalog główny:

#### src

pliki nagłówkowe modułów programu  
 pliki z kodem źródłowym modułów

#### bin

cellsim

#### in

pliki ustawień  
 pliki map

#### out

pliki png z zapisami plansz

plik Makefile

## 5. Etapy pracy symulatora

Po wczytaniu danych wejściowych funkcja sterująca `main()` w celu realizacji zadania wykorzysta kolejno następujące moduły:

1. Alokator pamięci (`alloc.h`)
2. Nadanie danych domyślnych (`alloc.h`)
3. Iteracja pokoleń (`gen.h`)
  - a) Generacja pokolenia (`gen.h`)
    - Szukanie sąsiedztwa (`nbr.h`)
    - Badanie przekroczenia tablicy (`life.h`)
    - Sprawdzanie życia (`life.h`)
    - Zastąpienie danych starego pokolenia (`gen.h`)
  - b) Zapisanie pokolenia do pliku (`nbr.h`)
4. Czyszczenie (`alloc.h`)

## 6. Dane wyjściowe

### 6.1. Pliki z zapisem

Generowane jest pliki z zapisem planszy wejściowej i generacji plansz występujący podany interwał czasowy. Ilość wygenerowanych plików była dana.

Pliki przedstawiają planszę zawierającą piksele będące reprezentacją żywych komórek; martwe nie są wyszczególnione. Wielkość pikseli i planszy jest wyskalowana.

Pliki typu png są zapisywane do uprzednio wyczyszczonego folderu `out`. Szablon nazw pliku jest opisany w punkcie 2.1. .





## 6.2. Generowanie plansz na żywo

Jeśli taka opcja jest uruchomiona, to oprócz standardowego generowania plików z zapisem zacznie się wyświetlać symulacja. Ekran konsoli będzie na bieżąco odświeżany i co pewien interwał czasowy będą się wyświetlać kolejne pokolenia automatu.

Wielkość planszy jest zaznaczona symbolami “|” i “-”. Żywe komórki są reprezentowane przez symbol “#”.

