

Automat komórkowy w języku C

1 Wstęp

Automat komórkowy jest systemem składającym się z pojedynczych komórek sąsiadujących ze sobą. Każda komórka przyjmuje jeden ze skończonego zbioru stanów. Komórka zmienia swój stan na podstawie przyjętych założeń, opierających się na stanie komórek z nią sąsiadujących. Takim przykładem automatu komórkowego jest gra w życie. Jej zasady to: Jeśli żywa komórka ma 2 lub 3 żywych sąsiadów to pozostaje żywa, w przeciwnym wypadku umiera. Jeśli martwa komórka ma 3 żywych sąsiadów to ożywa, inaczej pozostaje martwa. W naszej grze wykorzystywane jest sąsiedztwo Moore'a, które zakłada sąsiedztwo 8 komórek wokół komórki badanej.

2 Kompilacja kodu oraz wywołanie programu

W naszym projekcie przyjęliśmy, że przy wywoływaniu programu podawany jest plik o przykładowym wyglądzie:

```
3 4
1 0 0
1 1 1
1 1 1
0 0 0
```

W pierwszym wierszu pliku mamy wymiary planszy. Liczba 3 jest liczbą kolumn, natomiast liczby 4 jest liczbą wierszy naszej tablicy. Następne wiersze to już sama plansza, w której 0 reprezentują komórki martwe, a 1 komórki żywe. Aby wywołać program otwieramy terminal, a następnie wchodzimy w folder, w którym program się znajduje. Do terminalu wpisujemy komendę *make*, która korzystając z reguł zawartych w pliku Makefile kompiluje nasz kod oraz tworzy program *GameofLife*. Następnie wywołujemy program z wybranym przez nas plikiem, np.

```
./GameofLife twojplikzdanyimi.txt.
```

Program przechowuje planszę w tablicy dwuwymiarowej. Tworzone są 2 tablice. Jedna z nich używana jest do wprowadzania zmian w generacji, a druga służy do porównywania sąsiedztwa komórki w celu wprowadzenia zmian w pierwszej tablicy.

3 Funkcjonowanie programu

Program podzielono na dwa pliki *main.c* i *funkcje.c* oraz na plik nagłówkowy *funkcje.h*. Pierwszy z plików odpowiada za otwarcie pliku oraz wywołanie funkcji zawartych w drugim pliku. Kolejne etapy działania programu:

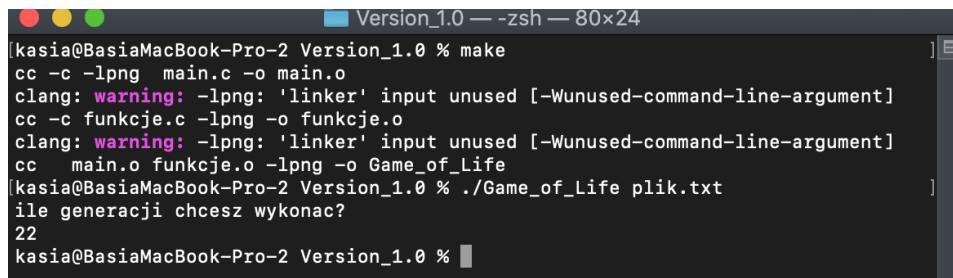
1. Pobranie danych z pliku i zapisanie ich do tablic.
2. Spytańie o ilość generacji i zapisanie tej danej.
3. Wygenerowanie nazwy pliku do generacji.
4. Porównanie komórek i wygenerowanie nowej planszy.
5. Utworzenie pliku *.png z wygenerowaną uprzednio nazwą i zapisanie do niego planszy w postaci komórka żywa - kolor biały, komórka martwa - kolor czarny.

Plik Makefile, o którym była mowa wcześniej zawiera 2 reguły, które kompilują kod zawarty w w plikach *main.c* oraz *funkcje.c* z flagą *-lpng*, aby dołączyć bibliotekę *libpng*. Korzystamy również z utworzonego przez nas pliku nagłówkowego "funkcje.h".

Wynikiem działania programu są pliki nazwie *Gen[x].png* , gdzie *x* jest numerem generacji.

4 Testy

Na początku przechodzimy do folderu *Version_{1.0}*, w którym znajduje się nasz program. W terminal wprowadzamy komendę *make*, która pozwoli przekompilować program. Program wywoływany jest za pomocą: *GameofLife*, dodatkowo należy po *./GameofLife* dodać nazwę pliku, na podstawie którego mają być generowane plansze. Nie podanie pliku lub podanie pliku, który nie istnieje spowoduje błąd i nasz program zakończy pracę. Jeśli program wywołamy prawidłowo spyta on nas ile planszy chcemy wygenerować.



```
Version_1.0 — -zsh — 80x24
[kasia@BasiaMacBook-Pro-2 Version_1.0 % make
cc -c -lpng main.c -o main.o
clang: warning: -lpng: 'linker' input unused [-Wunused-command-line-argument]
cc -c funkcje.c -lpng -o funkcje.o
clang: warning: -lpng: 'linker' input unused [-Wunused-command-line-argument]
cc main.o funkcje.o -lpng -o Game_of_Life
[kasia@BasiaMacBook-Pro-2 Version_1.0 % ./Game_of_Life plik.txt
ile generacji chcesz wykonac?
22
[kasia@BasiaMacBook-Pro-2 Version_1.0 % ]
```

Rysunek 1: użycie *make* i wywoływanie programu

Możemy potem łatwo zobaczyć, że program wygenerował pliki (w tym przykładzie 22). Dodatkowo na *Rysunku2* widać pliki z rozszerzeniem *.o, które stworzył *Makefile*.

```

kasia@BasiaMacBook-Pro-2 Version_1.0 % ls
Game_of_Life  Gen_15.png  Gen_21.png  Gen_8.png  main.c
Gen_1.png     Gen_16.png  Gen_22.png  Gen_9.png  main.o
Gen_10.png    Gen_17.png  Gen_3.png   Makefile   plik.txt
Gen_11.png    Gen_18.png  Gen_4.png   a.out
Gen_12.png    Gen_19.png  Gen_5.png   funkcje.c
Gen_13.png    Gen_2.png   Gen_6.png   funkcje.h
Gen_14.png    Gen_20.png  Gen_7.png   funkcje.o
kasia@BasiaMacBook-Pro-2 Version_1.0 %

```

Rysunek 2: utworzone pliki

Na koniec możemy wyczyścić nasz folder z wygenerowanych plasz i plików wykonywalnych (*.o) za pomocą komendy *make clean*.

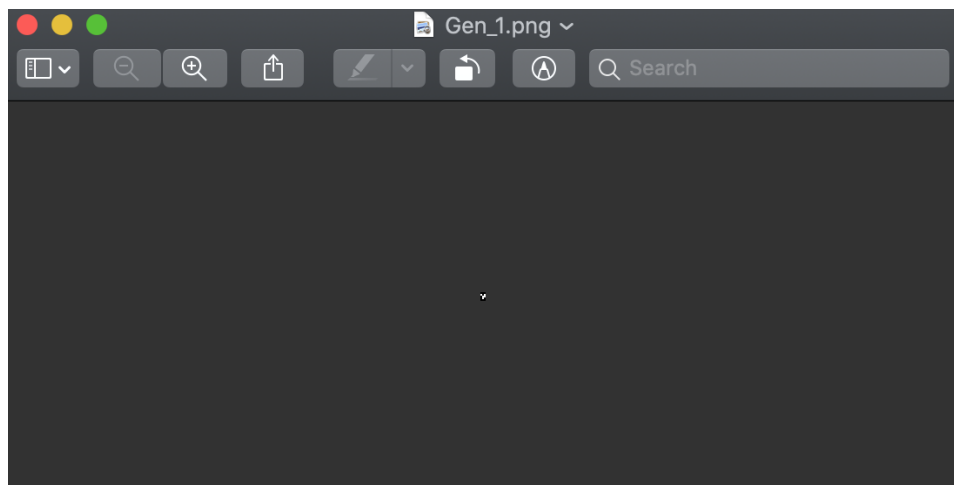
```

kasia@BasiaMacBook-Pro-2 Version_1.0 % make clean
rm *.o
rm *.png
kasia@BasiaMacBook-Pro-2 Version_1.0 % ls
Game_of_Life  a.out      funkcje.h  plik.txt
Makefile      funkcje.c  main.c
kasia@BasiaMacBook-Pro-2 Version_1.0 %

```

Rysunek 3: *make clean*

5 Pliki png



Rysunek 4: plik *png*

Na Rysunku 4 można zauważyć, że wygenerowany plik *png* jest bardzo mały, z tego względu trzeba oglądać go w powiększeniu (Rysunek 5).



Rysunek 5: Plik *png* w powiększeniu

6 Napotkane problemy

Trudności w projekcie można było podzielić na dwie części: programistyczne i środowiskowe. Pierwsze dotyczyły samego napisania kodu wykonującego schemat gry w życie i dający wynik w formacie **.png*. Drugie natomiast miały związek z obsługą Githuba, wyborem środowiska programistycznego, instalacją i konfiguracją LaTeXa. Istotą trudności projektu według mnie nie była część programistyczna, ponieważ tutaj mieliśmy już potrzebne narzędzia i tylko musieliśmy ich użyć. Natomiast druga część wymagała od nas odrobiny samodzielności i umiejętności radzenia sobie z nowymi przeszkodami.