Sprawozdanie końcowe

`GameOfLife`: Gra w życie Johna Conwaya

Aleksandra Michalska, Natalia Olszewska

05.04.2021

Spis treści

Część I
Informacje ogólne

1 O dokumencie

Dokument jest podsumowaniem projektu *GameOfLife: Gra w życie Johna Conwaya*. Koresponduje z dokumentami *"Specyfikacja funkcjonalna automatu komórkowego"* oraz *"Specyfikacja implementacyjna automatu komórkowego"*.

2 O programie

Program jest implementacją "Gry w Życie" Johna Conwaya. Został rozszerzony o dodatkowe zasady obliczania sąsiedztwa oraz rodzaje światów.

3 Środowisko implementacyjne i funkcjonalne

Program został zaimplementowany w języku programowania **C** w systemie operacyjnym **Linux** (dystrybucja Ubuntu). W celu kontroli wersji zostały użyte **GitHub** (utworzenie wstępnej wersji projektu) oraz **Projektor EE**, będący systemem kontroli wersji Politechniki Warszawskiej (utoworzenie ostatecznej wersji projektu).

Pozostałymi użytymi narzędziami były:

- **ImageMagick** - otwieranie zapisanych przez użytkownika obrazów
- **Valgrind** - przeprowadzenie testów wycieków pamięci
- **Vim** - podstawowa edycja kodu
- **Evince** - otwieranie dokumentacji plików o rozszerzeniu **pdf**
- **LaTeX** - tworzenie dokumentacji
- **Make** - utworzenie pliku **Makefile** i wywołanie programu

Część II
Funkcjonalności programu

4 Uruchomienie

Program można uruchomić korzystając z samodzielnie dobranych parametrów podstawowych i dodatkowych (patrz punkty 4.1.1 oraz 4.1.2) lub z gotowych poleceń zaimplementowanych w **Makefile** (`make run_sbs`, `make run_fast`, `make no_save`, `make after_change`).

4.1 Parametry wejściowe

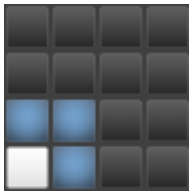
4.1.1 Parametry podstawowe

Z powodu kwestii implementacyjnych, sposób użycia następujących parametrów uległ zmianie:

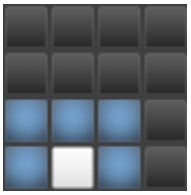
- `-in filein.txt` został zastąpiony `--in filein.txt`
- `-out fileout.txt` został zastąpiony `--out fileout.txt`
- `-how(Ms || Mf || Ns || Nf)` został zastąpiony `--how(Ms || Mf || Ns || Nf || Ps || Pf)`, przy czym:
  - `s` "sphere world" pozostaje bez zmian (patrz *"Specyfikacja funkcjonalna automatu komórkowego"*)
  - `f` "flat world" został zmodfikowany. W tym świecie, dla komórek:
    - na rogach, uznawanych jest **3** sąsiadów (patrz *Rysunek 1a*),
    - na obrzeżach, uznawanych jest **5** sąsiadów (patrz *Rysunek 1b*),
    - w środku, uznawanych jest **8** sąsiadów (patrz Rysunek *1c*).

    Zasady umierania i powracania do życia, w trybach podstawowych, pozostają takie same.

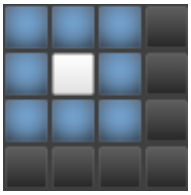
  Nowe rodzaje ustalania sąsiedztwa tj. `Pf` oraz `Ps`, zosatły opisane w punkcie 4.3.3.



(a) Komórka w rogu



(b) Komórka na brzegu



(c) Komórka w środku

Rysunek 1: Rozmieszczenie sąsiadów dla poszczególnych komórek w płaskim świecie

- `-s(o2 || f2)` został wycofany z parametrów obowiązkowych (patrz punkt 4.1.2)

Sposób użycia pozostałych parametrów pozostaje niezmieniony (patrz *"Specyfikacja funkcjonalna automatu komórkowego"*).

Przykładowe poprawne parametry wejściowe:

```
./game --in plikin.txt --out plikout.txt -n 20 -m sbs --how Ps
```

4.1.2 Parametry dodatkowe

Użytkownik ma możliwość skorzystania z dodatkowych, nieobowiązkowych parametrów:

1. Przy kompilacji programu:
   - `-DIN='"path_to_dir_input"'` zmiana ścieżki do katalogu, z którego pobierane będą pliki wejściowe
     (domyślny katalog: `'"Generacje_Wejściowe"'`)
   - `-DOUT='"path_to_dir_output"'` zmiana ścieżki do katalogu, w którym zapisywane będą pliki wyjściowe
     (domyślny katalog: `'"Generacje_Wyjściowe"'`)
   - `-DSIZE=10` zmiana rozmiaru boku jednej komórki (w pikselach) w generowanym obrazie (domyślny rozmiar: 40)
   - `-DHOW_FAST=200000` zmiana odstępu czasu (w mikrosekundach) pomiędzy wyświetlaniem kolejnych generacji w trybie `FAST` (domyślny czas: 210000)
2. Przy uruchamianiu programu:
   - `-s (o2 || f2)` sposób użycia pozostaje niezmieniony (patrz *"Specyfikacja funkcjonalna automatu komórkowego"*)

4.2 Obsługa podstawowa

Użytkownik uruchamia program przy pomocy gotowych poleceń zaimplementowanych w **Makefile**. Przykładowo, komenda:

```
make run_fast
```

oznacza polecenie:

```
./game --in starship.txt --out wunik.txt -s f5 -n 20 --how Ms -m
                                fast
```

Użytkownik ma także możliwość korzystania z samodzielnie dobranych parametrów (patrz punkt 4.1).

4.3 Obsługa szczegółowa

4.3.1 Tryb `SBS`

Tryb `SBS` umożliwia przechodzenie do kolejnych generacji po wciśnięciu dowolnego klawisza poza klawiszem ***e***. Jeżeli użytkownik poda więcej niż 1 klawisz, program pobierze tylko ostatni lub klawisz ***e***, jeżeli pojawił się wśród wybranych.

Przykładowo, podanie liter ***aaaaaab*** oznacza dla programu to samo, co podanie klawisza ***b***, natomiast podanie liter ***aaaeaab*** oznacza tyle co naciśnięcie klawisza ***e***.

Został także zmieniony sposób prześcia z trybu `SBS` do trybu `FAST`. Zamiast wybrania klawisza ***f*** użytkownik powinien wybrać klawisz ***e***.

4.3.2 Tryb `FAST`

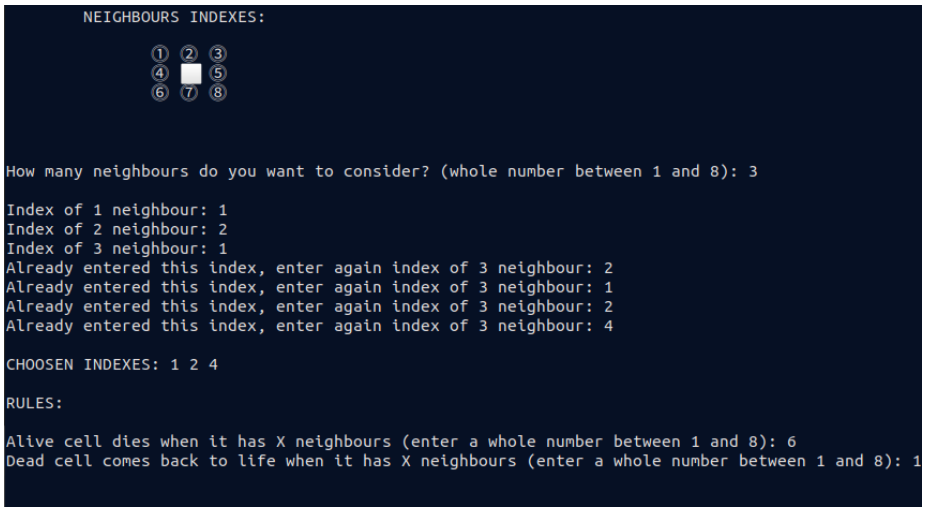
W tym trybie użytkownik nie wchodzi w interakcję z programem, kolejne generacje wyświetlane są automatycznie. Użytkownik nie ma możliwości zmiany tego trybu na inny ani przerwania pracy programu (poza wymuszeniem, które powoduje przerwanie pracy Makefile'a).

4.3.3 Rodzaj ustalania sąsiedztwa `PERSONAL`

W programie został zaimplementowany nowy sposób ustalania sąsiedztwa na podstawie preferencji użytkownika. Aby z niego skorzystać, należy przy wywołaniu użyć jednego z poniższych parametrów:

- `--how Ps` czyli własny sposób ustalania sąsiedztwa, świat działający jak sfera
- `--how Pf` czyli własne sposób ustalania sąsiedztwa, świat działający jak płaska plansza

W przypadku wybrania tego trybu, zostaje wyświetlony interface, w którym użytkownik wybiera jak chce ustalać sąsiedztwo:



Rysunek 2: Ustawianie własnych zasad sąsiedztwa

5 Dodatkowe funkcje

W celu ułatwienia korzystania z programu użytkownik ma możliwość skorzystania z komend pomocniczych:

- `make remove_out` umożliwia usunięcie wszystkich plików wyjściowych z katalogu z generacjami wyjściowymi
- `make remove_image` umożliwia usunięcie wszystkich obrazów z katalogu `Obrazy`

6 Wyniki działania programu

6.1 Wyświetlanie generacji w terminalu

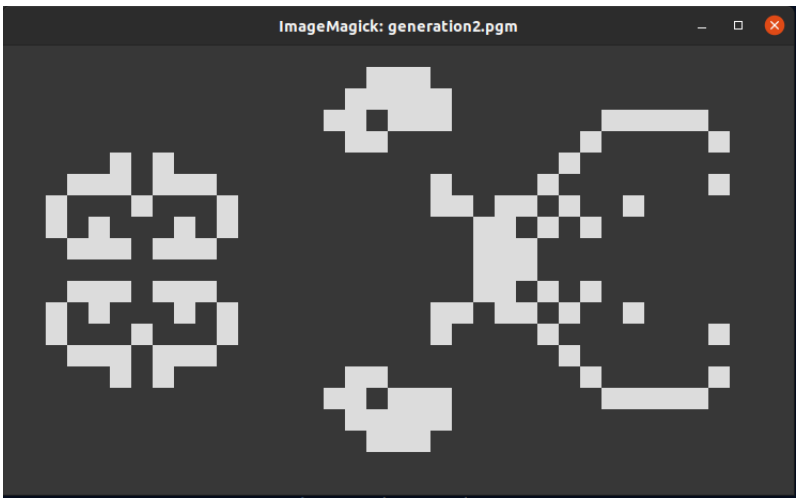
Kolejne generacje wyświetlne są w terminalu (patrz *Rysunek 3* i *Rysunek 4*). Wyświetlana jest także informacja o numerze obecnej generacji oraz liczbie żywych komórek.

Rysunek 3: Interface trybu `FAST`

Rysunek 4: Interface trybu `SBS`

6.2 Zapis plików graficznych

Wybrane przez użytkownika generacje można wyświetlić przy użyciu programu takiego jak ImageMagick (patrz *Rysunek 5*).



Rysunek 5: Wyświetlanie obrazu generacji przy pomocy programu **ImageMagick**

6.3 Zapis generacji wyjściowych

Ostatnia generacja zapisywana jest do pliku o nazwie podanej przez użytkownika przy wywołaniu programu (patrz punkt 4.1.1). Użytkownik ma możliwość użycia jej w programie jako generacji wejściowej (patrz *Rysunek 6*).

```
21 37
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0
0 0 0 0 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 1 0 0
0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 0 0 1 1 1 0 1 1 0 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

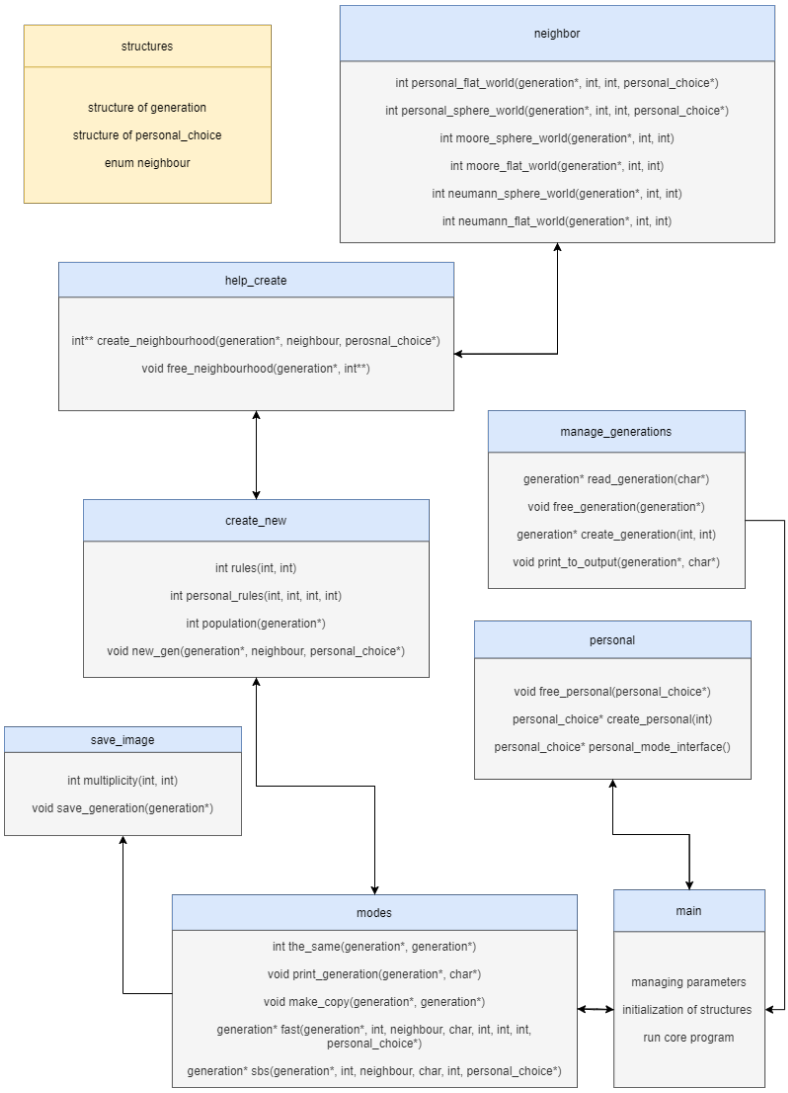
Rysunek 6: Plik tekstowy z generacją wyjściową

Część III
Implementacja programu

7 Informacje ogólne

7.1 Nowy diagram modułów

W celu zapewnienia większej czytelności kodu programu oraz dodania nowych funkcjonalności, został ponownie sporządzony diagram modułów (patrz *Rysunek 7*).



Rysunek 7: Końcowy diagram modułów

7.2 Informacje ogólne o zmianach w modułach

Struktury, które miały być zdefiniowane w module **main**, zostały przeniesione do osobnego modułu pełniącego rolę pliku nagłówkowego.

Dodatkowo została dodana nowa struktura **personal_choice**, używana do obsługi własnego trybu sąsiedztwa.

Podobnie, obsługę plików przeniesiono do nowego modułu **manage_generations**, natomiast w module **main** jest ona tylko wywoływana.

Został wprowadzony nowy typ sąsiedztwa, a co za tym idzie pojawił się kolejny moduł **personal**, który obsługuje jego podstawowe funkcje.

Z powodu używania zmiennych strukturalnych jako argumentów wielu funkcji, **structures** jest wykorzystywane we wszystkich modułach.

8 Informacje szczegółowe o modułach

8.1 main

Moduł zawiera tylko jedną funkcję **main**. Skupia się on wyłącznie na obsłudze parametrów wsadowych, tworzeniu inicjalizacji głównych zmiennych oraz uruchamianiu kluczowych funkcji z innych modułów. Zawiera wszelkie potrzebne obsługi błędów.

Podstawową zmianą względem *"Specyfikacji implementacyjnej automatu komórkowego"* jest przeniesienie obsługi plików oraz definiowanie struktur do osobnych modułów.

8.2 personal

Cały moduł jest względnie nowy wobec poprzednich specyfikacji. Powstał na potrzeby kontrolowania nowego trybu sąsiedztwa `PERSONAL`.

Moduł zawiera trzy funkcje (patrz *Tabela 1*).

8.3 modes

Funkcjonowanie tego modułu nie zmieniło się, jednak parametry funkcji trybów zostały rozszerzone. Ponadto dodano trzy nowe funkcje wspomagające działanie trybów (patrz *Tabela 2*).

8.4 save_image

Ze względu na łatwiejszą implementację, wbrew pierwotnym założeniom, obraz generacji nie jest zapisywany do pliku o rozszerzeniu **bmp**, a **pgm**. Została dodana także funkcja pomocnicza licząca wielokrotność liczby (patrz *Tabela 3*).

8.5 manage_generations

Nowopowstały moduł w celu odciążenia **main**, zawiera podstawowe funkcje obsługujące zmienne typu `generation` (patrz *Tabela 4*).

8.6 create_new

Moduł został rozszerzony o nową funkcję zasad, dla nowego trybu sąsiedztwa. Dodano także funkcję liczącą populację (patrz *Tabela 5*).

8.7 help_create

Do modułu dodano jedną funkcję czyszczącą dwuwymiarową tablicę sąsiedztwa. Natomiast funkcja tworząca sąsiedztwo została wzbogadzona o jeden argument dla spersonalizowanego typu sąsiedztwa (patrz *Tabela 6*).

8.8 neighbor

Oprócz trybów sąsiedztwa opisanych w specyfikacjach, zostały dodane dwa nowe. Było to spowodowane dodaniem opcji spersonalizowania reguł sąsiedztwa (patrz *Tabela 7*).

8.9 stuctures

Struktury zostały wyodrębnione w osobny modul. Dodatkowo utowrzona została nowa struktura **personal_choice**, w celu obsługi spersonalizowanego trybu sąsiedztwa.

Nowa struktura zawiera cztery parametry:
- **dies** typu **int**, określająca od jakiej liczby sąsiadów komórka ma umrzeć.
- **comes_back** typu **int**, określająca od jakiej liczby sąsiadów komórka ma ożyć.
- **personal_neighbours** typu **int**, określająca maksymalną liczbę sąsiadów komórki.
- **neighbours_index** typu **int***, zapamiętująca indeksy sąsiadów komórki.

9 Dodatkowe implementacje

9.1 Makefile

Oprócz podstawowych funkcji został utworzony **Makefile** w celu łatwiejszego uruchamiania programu.

Zawierają się w nim wszelkie możliwe rodzaje kompilacji oraz uruchomienia. Przy zmianie tylko argumentów wejściowych uruchomienia programu, można wymieniać rodzaje kompilacji. Po każdym uruchomieniu następuje usunięcie pliku wynikowego.

Przykład podstawowego pliku wynikowego:

```
game:
$(CC) -g Moduly/main.c Moduly/modes.c Moduly/save_image.c
Moduly/manage_generations.c Moduly/create_new.c
Moduly/help_create.c Moduly/neighbor.c Moduly/personal.c
-Wall -pedantic -o game
```

Przykład implementacji uruchomienia w trybie sbs:

```
run_sbs: game
./change -in plikin.txt -out plikout.txt -n 20 -how Ps -m sbs
rm -f change
```

Dodatkowo zostały zaimplementowane opcje wyczyszczenia zawartości katalogów `'"Generacje_Wyjsciowe"'` oraz `'"Obrazy"'`:

```
remove_out:
rm Generacje_Wyjsciowe/*.txt
remove_image:
rm Obrazy/generation*
```

Ponadto w fazie testowania zostały napisane formuły uruchomienia poszczególnych tesów, w celu sprawdzenia poprawności otrzymywanych wyników.

9.2 Makra

W celu zwiększenia możliwości personalizacji wyników programu, napisane zostały zmienne w postaci makr. Dotyczą one takich elementów jak ścieżki do plików z generacjami wejściowymi i wyjściowymi, rozmiar boku komórki przy zapisie obrazu oraz szybkość wyświetlania generacji podczas uruchomienia trybu `FAST`.

10 Podsumowanie zmian

Do każdego modułu zostały wprowadzone zmiany. Niektóre powstały od nowa (**manage_generations**, **structures**, **personal**), a zdecydowana większość została rozszerzona o dodatkowe funkcje. Został także dodany **Makefile** w celu ułatwienia uruchamiania. Ponadto, w celu możliwości personalizacji działania programu, został dodany nowy tryb sąsiedztwa `PERSONAL` oraz makra, które przy kompilacji programu pozwalają nam zmienić niektóre parametry. Ze względu na ułatwienia implementacyjne zapis generacji odbywa się do pliku graficznego o rozszerzeniu **pgm**.

11 Tabele

Tabela 1: Tablica funkcji modułu **personal**

typ funkcji	nazwa funkcji	argumenty funkcji	cel funkcji
void	free_personal	personal_choice*	zwolnienie pamięci zmiennej typu **personal**
personal_choice*	create_personal	int	inicjalizacja zmiennej typu **personal**
perosnal_choice*	personal_mode_interface		przypisanie wartości zmiennej typu **personal** zgodnie z danymi pobranymi od użytkownika

Tabela 2: Tablica funkcji modułu **modes**

typ funkcji	nazwa funkcji	argumenty funkcji	cel funkcji
void	print_generation	generation* char*	wyświetlenie pojedyńczej generacji na ekran
int	the_same	generation* generation*	sprawdzenie czy dwie generacje wyglądają tak samo
void	make_copy	generation* generation*	zapisanie generacji jednej zmiennej do drugiej
generation*	fast	generation* int neighbour char int int int personal_choice*	uruchomienie trybu fast
generation*	sbs	generation* int neighbour char int personal_choice*	uruchomienie trybu sbs

Tabela 3: Tablica funkcji modułu **save_image**

typ funkcji	nazwa funkcji	argumenty funkcji	cel funkcji
int	mupltiplicity	int int	liczenie jaka to wielokrotność danej liczby
void	save_generation	generation*	zapisanie generacji do pliku graficznego o rozszerzeniu **pgm**

Tabela 4: Tablica funkcji modułu **manage_generations**

typ funkcji	nazwa funkcji	argumenty funkcji	cel funkcji
generation*	read_generation	char*	przypisanie wartości zmiennej typu **generation** zgodnie z danymi z pliku
void	free_generation	generation*	zwolnienie pamięci przypisanej dla zmiennej typu **generation**
generation*	create_generation	int int	inicjalizacja zmiennej typu **generation**
void	print_to_output	generation* char*	zapisanie tablicy ostatniej generacji do pliku tekstowego

Tabela 5: Tablica funkcji modułu **create_new**

typ funkcji	nazwa funkcji	argumenty funkcji	cel funkcji
int	rules	int int	określenie stanu komórki dla nowej generacji w podstawowym uruchomieniu
int	personal_rules	int int int int	określenie stanu komórki dla nowej generacji w spersonalizowanym uruchomieniu
int	population	generation*	zliczenie populacji dla pojedyńczej generacji
void	new_gen	generation* neighbour personal_choice*	utworzenie nowej generacji

Tabela 6: Tablica funkcji modułu **help_create**

typ funkcji	nazwa funkcji	argumenty funkcji	cel funkcji
int**	create_neighbourhood	generation* neighbour personal_choice*	tworzenie tablicy sąsiedztwa
void	free_neighbourhood	generation* int**	zwolnienie pamięci zarezerwowanej dla tablicy sąsiedztwa

Tabela 7: Tablica funkcji modułu **neighbor**

typ funkcji	nazwa funkcji	argumenty funkcji	cel funkcji
int	personal_flat_world	int int int personal_choice*	liczenie sąsiadów komórki dla spersonalizowanego trybu sąsiedztwa w płaskim świecie
int	personal_sphere_world	int int int personal_choice*	liczenie sąsiadów komórki dla spersonalizowanego trybu sąsiedztwa w sferycznym świecie
int	moore_sphere_world	generation* int int	liczenie sąsiadów komórki dla podstawowego trybu sąsiedztwa Moore'a w sferycznym świecie
int	moore_flat_world	generation* int int	liczenie sąsiadów komórki dla podstawowego trybu sąsiedztwa Moore'a w płaskim świecie
int	neumann_sphere_world	generation* int int	liczenie sąsiadów komórki dla podstawowego trybu sąsiedztwa Neumanna w sferycznym świecie
int	neumann_flat_world	generation* int int	liczenie sąsiadów komórki dla podstawowego trybu sąsiedztwa Neumanna w płaskim świecie

Część IV
Sprawozdanie Testów

12 Cel testów

W fazie testowania szczególna uwaga została poświęcona obsłudze błędów pamięci, parametrów wejściowych oraz zmiennych otrzymanych od użytkownika poprzez interface trybu `PERSONAL`. Celem było udoskonalenie programu oraz sprawienie by był możliwie najlepiej przystosowany do potrzeb użytkownika.

13 Plan testów

Oprócz wcześniej zadeklarowanych testów (patrz *"Specyfikacja implementacyjna automatu komórkowego"*) należało dodać testy dodatkowe. Było to spowodowane zmianami i rozszerzeniami w modułach programu.

Do dodatkowych testów zaliczają się:

- sprawdzenie poprawności obsługi parametrów uruchomienia interface'u,
- zapis obrazów mimo zakończenia wyświetlania nowych generacji, w skutek sytuacji, gdy kolejne generacje są identyczne,
- obsługa wprowadzenia zbyt dużej liczby znaków dla trybu `SBS`.

14 Wyniki

Zgodnie z przewidywaniami, testy wypadły pomyślnie. W momentach implementacji występowały błędy, które w miare szybko zostawały rozwiązywane. Także w czasie testów ostatecznych, wszelkie wyniki były przewidywalne i prawidłowe.

Sprawdzenie stanu pamięci po uruchomieniu programu (patrz *Rysunek 8*).

```
==27729==
==27729== HEAP SUMMARY:
==27729==     in use at exit: 0 bytes in 0 blocks
==27729==   total heap usage: 513 allocs, 513 frees, 112,028 bytes allocated
==27729==
==27729== All heap blocks were freed -- no leaks are possible
==27729==
==27729== For lists of detected and suppressed errors, rerun with: -s
==27729== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Rysunek 8: Wynik działania programu **Valgrind**

Sprawdzenie obsługi błędów parametrów wejściowych (patrz *Rysunek 9*).

```
./game --in starship.txt --out wynik.txt -n 30 --how Ms -m Maryla_Rodowicz

        Parameter Error

make: *** [Makefile:72: test4.6] Błąd 1
```

Rysunek 9: Podanie nieistniejącej wartości parametru

Sprawdzenie obsługi błędów dotyczących plików wejściowych (patrz *Rysunek 10*).

```
./game --in blad_kolumn.txt --out wynik.txt -n 30 --how Ns -m sbs

        FileError

make: *** [Makefile:48: test2] Błąd 1
```

Rysunek 10: Próba użycia niewłaściwego pliku wejściowego

Podczas korzystania z trybu sąsiedztwa `PERSONAL`, istnieje możliwość wprowadzenia błednych danych.

Jednym z nich jest powtórzenie numeru indeksu. W tym momencie program przypomni użytkownikowi, że taki indeks już się pojawił i należy wybrać inny (patrz *Rysunek 11*).

```
        NEIGHBOURS INDEXES:

              ① ② ③
              ④ ■ ⑤
              ⑥ ⑦ ⑧


How many neighbours do you want to consider? (whole number between 1 and 8): 5

Index of 1 neighbour: 1
Index of 2 neighbour: 1
Already entered this index, enter again index of 2 neighbour: 2
```

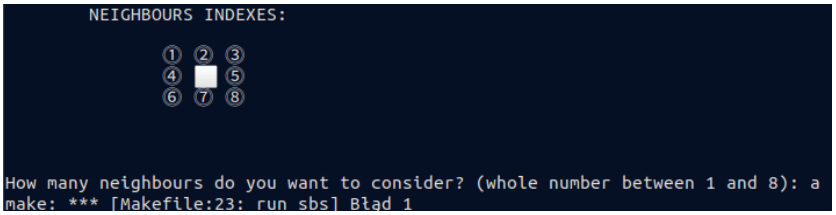
Rysunek 11: Ponowny wybór tego samego indeksu

Podobnie, użytkownik może podać liczbę wykraczającą poza dany przedział. Zostanie on wówczas o tym poinformowany, a program zakończy pracę (patrz *Rysunek 12*).

```
        NEIGHBOURS INDEXES:

              ① ② ③
              ④ ■ ⑤
              ⑥ ⑦ ⑧


How many neighbours do you want to consider? (whole number between 1 and 8): -5

Number out of range. Exiting.

make: *** [Makefile:23: run_sbs] Błąd 1
```

Rysunek 12: Niepoprawne dane wykraczające poza akceptowany przedział

Natomiast, gdy użytkownik poda literę zamiast liczby, program zakończy pracę bez ostrzeżenia (patrz *Rysunek 13*).



Rysunek 13: Wprowadzenie zmiennej o niepoprawnym typie

15 Zmiany spowodowane wynikami testów

W czasie tworzenia programu wystąpił tylko jeden moment, w którym trzeba było zmienić jego działanie, aby naprawić błąd. Miał on miejsce w czasie pobierania argumentu w trybie `SBS`. Należało pobierać wszelkie dodatkowe znaki, aby nie pozostawały w buforze wejścia. W taki sposób ostateczny program przyjmuje tylko ostatni znak przed ***ENTER*** lub znak ***e***, jeśli wystąpił w trakcie wpisywania.

16 Wnioski na podstawie testów

Testy wypadły pomyślnie, jednak nie oznacza to, że programu nie można bardziej udoskonalić. Program działa zgodnie z początkowymi założeniami, a wszelkie podstawowe błędy są obsługiwane.

Część V
Spostrzeżenia i uwagi końcowe

17 Osiągnięcie celu projektu

Program działa prawidłowo, udało się także poprawnie zaimplementować wszystkie rozszerzenia. Można zatem uznać, że cel projektu został osiągnięty.

18 Elementy niedoskonałe i możliwe błędy

Pomimo dobrych funkcjonalności oraz pomyślnie przeprowadzonych testów, program nie został maksymalnie zabezpieczony. W przypadku zmiany nazw katalogów, program nie zabezpiecza błędów. Informuje o naruszeniu pamięci, co skutkuje nagłym zakończeniem jego pracy.

Podobnie, komunikaty o błędach nie są sprecyzowane, co może utrudnić użytkownikowi zrozumienie niepowodzenia.

Odczuwalnym ograniczeniem jest brak możliwości szybszego zakończenia pracy programu, bez konieczności przerwania wykonywania pracy Makefile'a.

19 Ograniczenia programu

Program ma pewne ograniczenia, wynikające z możliwości użytego sprzętu. Jeśli urządzenie ma mało dostępnej pamięci, poprawne działanie programu może być utrudnione a nawet niemożliwe.

Ponadto, zapis obrazów możliwy jest tylko do pliku o rozszerzeniu bitowym **pgm**, którego podgląd może być utrudniony, ze względu na możliwość konieczności instalowania dodatkowych narzędzi.

Uwzględnione zostały także następujące restrykcje dotyczące rozmiaru generacji:

- maksymalna liczba kolumn wynosi 50, a wierszy 40 (przez ograniczone wymiary okna konsoli)
- minimalna liczba kolumn i wierszy wynosi 4 (ze względu na logikę zagadnienia)

20 Podsumowanie współpracy

Współpraca przebiegła bez komplikacji. Obie strony były bardzo zaangażowane, można było zauważyć dużą chęć udoskonalenia projektu.