RAPORT

Paweł Skiba, Jakub Świder

14 kwietnia 2019

Spis treści

1 Wstęp **2**

2 Ostateczny projekt modułów **2**

3 Opis modyfikacji modułów **3**
3.1 Main 3
3.2 Game 4
3.3 ReadFile 7
3.4 WriteFile 7
3.5 PNG 9

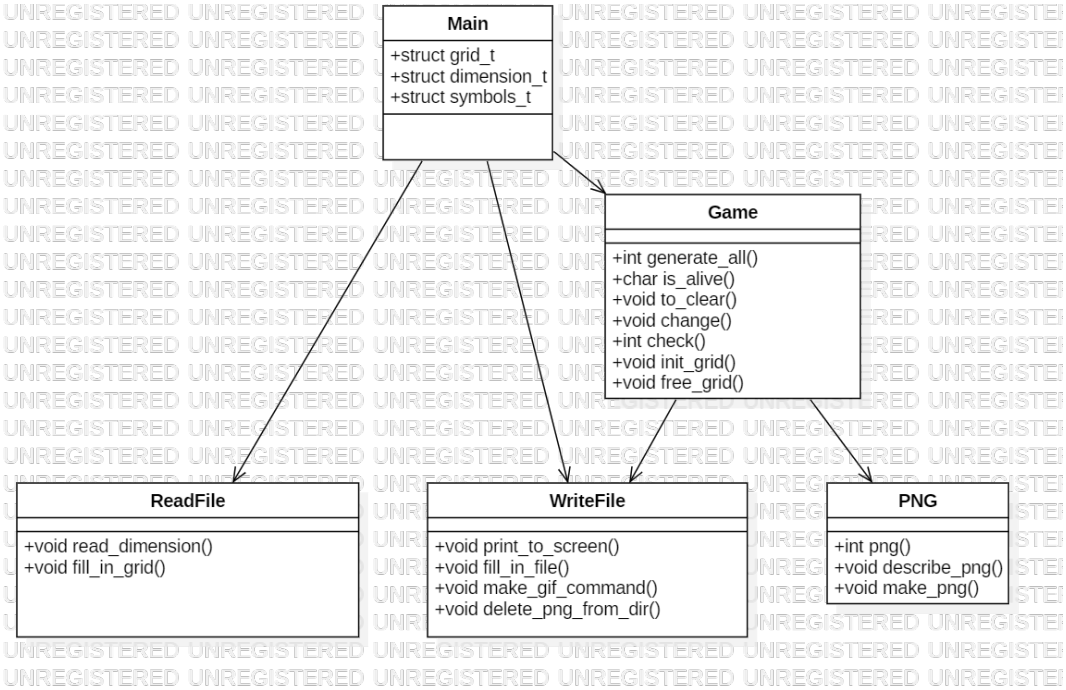
4 Prezentacja działania **10**
4.1 Przykładowe wywołanie 10
4.2 Plik wejściowy 11
4.3 Wynik wywołania 12

5 Podsumowanie testów modułów **12**
5.1 Moduł ReadFile - test_fill_in_grid.c 13
5.2 Moduł Game - test_is_alive.c 14
5.3 Moduł WriteFile - test_print_to_screen.c 15
5.4 Moduł PNG - test_png.c 16
5.5 Pamięć - valgrind 17

1 Wstęp

W związku z wieloma zmianami, które zostały wprowadzone podczas tworzenia projektu, w raporcie zostaną przedstawione ostateczne założenie projektowe. Również zostanie przedstawiona prezentacja działania oraz przestawione testy, które były niezbędne w trakcie tworzenia programu. Podczas tworzenia kodu źródłowego, wiele opcji, które założyliśmy okazały się błędne, co spowodowało modyfikację projektu. Najbardziej zostały zmodyfikowane sygnatury funkcji, które przyjmowały zbyt mało argumentów. W wielu przypadkach nie uwzględniliśmy struktur, które przechowywują rozmiar oraz struktur, które przechowywujących symbole wprowadzene przez użytkownika za pomocą flag. Dodatkowo nie zakładaliśmy stworzenia Modułu PNG, który dodatkowo wykonaliśmy. Również został rozszerzony Moduł WriteFile o dodatkową możliwość tworzenia plików GIF. Wszystkie te zmiany zostaną zaprezentowane w poniższym raporcie.

2 Ostateczny projekt modułów



Rysunek 1: Diagram modułów

3 Opis modyfikacji modułów

3.1 Main

1. **Funkcja main()**

   Poprzednio:
   int main(int argc, char *argv[]);

   Aktualnie:
   int main(int argc, char *argv[]);

   Opis:
   Moduł Main posiada tylko jedną funkcję - main(), która jest odpowiedzialna za sterowanie programem. Obsługuje ona argumenty wejściowe, a następnie na ich podstawie uruchamia poszczególne moduły. Użytkownik może za pomocą opcji wprowadzić następujące argumenty:

   (a) -n: Określa liczbe generacji. Argumentem jest liczba całkowita dodatnia

   (b) -i: Określa nazwę tesktowego pliku wejściowego, na podstawie, którego program rozpocznie generację plansz. Argumentem jest ciąg znaków zakończony '.txt'. Plik o takiej nazwie musi istnieć w folderze z programem

   (c) -o: Określa nazwę pliku tekstowego do którego będą zapisane wyniki działania programu. Argumentem jest ciąg znaków zakończony '.txt'. Jeśli plik o danej nazwie nie istnieje to zostanie on stworzony przez program. DOmyślnie jest to out.txt

   (d) -g: Określa nazwę pliku GIF, do którego zostaną zapisane wyniki działania programu. Argumentem jest ciąg znaków zakończony '.gif'. Jeśli plik o danej nazwie nie istnieje to zostanie on stworzony przez program. Domyślnie jest to out.gif

   (e) -l: Określa opóźnienie wyświetlania obrazów w pliku GIF. Argumentem jest liczba zmiennoprzecinkowa

   (f) -a: Określa symbol oznaczający komórki żywe w pliku wejściowym i pliku tekstowym, do którego zapisane będą wyniki programu. Argumentem jest jeden symbol. Domyślnie jest to 1

   (g) -d: Określa symbol oznaczający komórki martwe w pliku wejściowym i pliku tekstowym, do którego zapisane będą wyniki programu. Argumentem jest jeden symbol. Domyślnie jest to 0

(h) -w: Określa sposób w jaki program zapisze lub wyświetli wyniki działania. Argumentem może być liczba 0, 1 lub 2. Domyślnie jest to 2

- dla -w 0: Program wypisze wyniki działania na wyjście standardowe
- dla -w 1: Program wypisze wyniki działania do pliku tekstowego
- dla -w 2: Program stworzy dla każdej generacji plik w formacie PNG, a nsatępnie na ich podstawie stworzy plik w formacie GIF

Dostępna jest jeszcze opcja -h. Po jej zastosowaniu zostanie wyświetlony wbudowany help.

3.2 Game

1. **Funkcja generate_all()**

   Poprzednio:
   void generation(int n, grid_t * ptr, dimension_t * ptrd);

   Aktualnie:
   int generate_all(int n, int writeOpt, char *fileOut, dimension_t *dim, grid_t *main_grid, grid_t *util_grid, symbols_t * syms);

   Opis:
   Funckja ta jest uruchomiana z poziomu funkcji main() oraz przejmuje sterowanie nad grą. W porównaniu do pierwotnego podejścia, funkcja otrzymuje dużo więcej argumentów. Nie założyliśmy, że niezbędna będzie ilość generacji, wskaźnik na planszaę pomocniczą czy opcjonalną struktura ze znakami. Funkcja ta iteracyjnie wywołuje funkcję is_alive(), która implementuje zasady gry. Również z jej poziomu są wywoływane funkcje pomocniczne, takie jak to_clear(), change() czy check(). Funkcja generate_all() po zakończeniu działania zwraca liczbę wygenerowanych plansz, która służy do poprawnego działania funckji make_gif_command() i stworzenia pliku .gif

2. **Funkcja is_alive()**

   Poprzednio:
   bool is_alive(char *cell, grid_t *grid);

   Aktualnie:
   char is_alive(int i, int j, char ** ptr, symbols_t * syms);

   Opis:
   Został zmieniony typ funkcji z typu bool na typ char, został również rozszerzony zakres przyjmowanych argumentów. Niezbędne okazały się indeksy planszy oraz podwójny wskaźnik typu char. Funckcja działa na zasadzie przyjmowania w każdej generacji indeksów aktualnie badanej komórki. Iteruje po każdej komórce otaczającej dany element, zgodnie z zasadami gry i na ich podstawie zwraca oczekiwany znak.

3. **Funkcja to_clear()**

   Poprzednio:
   clear(grid_t *to_clear);

   Aktualnie:
   void to_clear(grid_t *toclear, dimension_t *dim, symbols_t * syms);

   Opis:
   Funkcja przyjmuje wskaźnik na strukturę z planszą i zapełnia ją komórkami martwymi na podstawie wymiarów (z dimension_t) i symboli oznaczających komórki martwe (z symbols_t). Poprzednia wersja funckji nie uwzględniała wymiarów planszy oraz możliwości wyboru przez użytkownika innych symboli komórek niż domyślne (0 i 1).

4. **Funckja change()**

   Poprzednio:
   Brak

   Aktualnie:
   void change(grid_t *first, grid_t *second, dimension_t *dim);

   Opis:
   Niezbędna okazała się funckja change(), która po każej generacji przyjmuje adres planszy na podstawie, której przebiegła ostatnia generacja oraz adres nowo wygenerowanej planszy. Następnie zapisuje znaki z drugiej planszy do pierwszej planszy, tak ay kolejna generacja przebiegła na podstawie nowo wygenerowanej planszy.

5. **Funckja check()**

   Poprzednio:
   Brak

   Aktualnie:
   int check(grid_t *check_1, grid_t *check_2, dimension_t *dim);

   Opis:
   Funckja po kadżej generacji przyjmuje adres planszy, na podstawie której przebiegła ostatnia generacja oraz adres nowo wygenerowanej planszy. Jeżeli wszystkie znaki w obu planszach są takie same program przerwie generowanie nowych plansz i wypisze odpowiedni komunikat, który wskaże od której generacji plansze są takie same.

6. **Funckja init_grid()**

   Poprzednio:
   Brak

   Aktualnie:
   void init_grid(grid_t * grid, dimension_t * dim, symbols_t * syms);

   Opis:
   Funkcja przyjmuje adres zadeklarowanej struktury z planszą, na podstawie wymiarów alokuje pamięć dla tej planszy oraz wypełnia ją komórkami martwymi.

7. **Funckja free_grid()**

   Poprzednio:
   Brak

   Aktualnie:
   void free_grid(grid_t grid, dimension_t dim);

   Opis:
   Funkcja przyjmuje adres struktury z planszą i zwalnia pamięć, która została wcześniej dla niej zaalokowana.

3.3 ReadFile

1. **Funkcja read_dimension()**

   Poprzednio:
   void read_file(char *filename, dimension_t *dim);

   Aktualnie:
   void read_dimension(char *filename, dimension_t *dim,
   symbols_t * syms);

   Opis:
   Funkcja read_dimension zasadniczo nie została zmieniona w kontekście sygnatury. Dodatkowo musi ona pobierać wskaźnik na strukturę symboli, dzięki której może bezbłędnie wykonać swoje zadanie. Funkcja zlicza wszystkie wiersze oraz kolumny znaków, które zostały zapisane w pliku tekstowym, a zarazem sprawdza poprawną zawartość pliku tekstowego, co odciąża funkcję fill_in_grid().

2. **Funkcja fill_in_grid()**

   Poprzednio:
   void fill_in_grid(char *filename, grid_t *grid_pointer);

   Aktualnie:
   void fill_in_grid(char *filename, grid_t *poiv, symbols_t * syms);

   Opis:
   Funckja fill_in_grid() podobnie jak funkcja read_dimension() została rozszerzona o wskaźnik na strukturę symboli. Odpowiada ona za wypełnienie planszy znakami, które są zawarte w pliku. Bardzo ważnym aspektem poprawnego funkcjonowania tej funkcji jest poprawność znaków w pliku oraz poprawne wprowadzenie ich, jako argumentów wywołania podczas uruchamiania programu.

3.4 WriteFile

1. **Funkcja print_to_screen()**

   Poprzednio:
   void print_to_screen(grid_t * ptr);

   Aktualnie:
   void print_to_screen(int n, grid_t *to_write, dimension_t *dim);

Opis:
Funkcja print_to_screen() ma na celu wyświetlenie informajci, po której generacji zostanie wyświetlona plansza, co spowodowało rozbudowę sygnatury o dodatkową zmienną typu int oraz bezpośrednio pod tą informacją zostaje wyświetlona plansza przedstawiająca stan gry po tej iteracji. Niezbędna okazała się również informacja na temat wymiarów, co spowodowało dopisanie wskaźnika na strukturę typu dimension_t.

2. **Funkcja fill_in_file()**

   Poprzednio:
   void write_grid(char *filename, grid_t * ptr);

   Aktualnie:
   void fill_in_file(int n, char *filename, grid_t *to_write,
   dimension_t *dim);

   Opis:
   Funkcja ta ma za zadanie zapisać każdą iterację do tego samego pliku. Otwiera ona plik i dopisuje do niego kolejną iterację nie usuwając poprzedniej, a następnie go zamyka. Sposób działania jest identyczny jak w funkcji print_to_screen().

3. **Funkcja make_gif_command()**

   Poprzednio:
   Brak

   Aktualnie:
   void make_gif_command(double delay, int gen_num,
   char *gifFilename);

   Opis:
   Funkcja przyjmuje wpisaną przez użytkownika wartość opóźnienia w pliku .gif, nazwę pliku .gif do jakiego zostanie zapsany wynik działania programu, oraz liczbę wygenerowanych plików .png, z ktorych zostanie stworzony plik .gif. Funkcja następnie wywołuje komendę 'convert' z pakietu 'imagemagick', która tworzy plik .gif.

4. **Funkcja delete_png_from_dir()**

   Poprzednio:
   Brak

   Aktualnie:
   void delete_png_from_dir(void);

Opis:
Funckja wywołuje serię komend z SO Linux, w wyniku których wszystkie pliki .png, będące pozostałością po poprzednim wywołaniu programu, zostają usunięte. Funkcja również stworzy folder Obrazy_PNG konieczny do prawidłowego działania programu, jeśli takowy nie istnieje.

3.5 PNG

1. **Funkcja png()**

   Poprzednio:
   Brak

   Aktualnie:
   void png(grid_t *util_grid, dimension_t *dim, symbols_t *syms, int n);

   Opis:
   Funkcja przyjmuje listę argumentów niezbędnych do stworzenia obrazu PNG. Jest ona odpowiedzialna za stworzenie nazwy pliku do którego będzie zapisywana generacja. Funckja wywołuje także w odpowiedniej kolejności pozostałe funkcje modułu PNG - describe_png(), a następnie make_png().

2. **Funkcja describe_png()**

   Poprzednio:
   Brak

   Aktualnie:
   void describe_png(grid_t *util_grid, dimension_t *dim,
   symbols_t *syms);

   Opis:
   Funkcja describe_png() jest odpowiedzialna za zaalokowanie pamięci potrzebnej do utworzenia tablicy dwuwymiarowej odpowiadającej pikselom. Następnie funkcja ta wypełnia zarezerwowaną pamięć na podstawie planszy util_grid, co sprawia, że plansza pikseli jest wypełniona odpowiednimi wartościami. Stan każdej komórki jest reprezentowany przez 100 pikseli reprezentujących kwadrat w formacie 10 x 10 pikseli.

3. **Funkcja make_png()**

   Poprzednio:
   Brak

   Aktualnie:
   void make_png(char *file_name);

   Opis:
   Funkcja make_png() tworzy i zapisuje obraz PNG na podstawie tablicy dwuwymiarowej, która została stworzona przez funkcję describe_png(). Wykorzystuje ona do tego niezbędne funkcje, które są zaimplementowane w bibliotece ¡png.h¿. Po wykonaniu tych operacji, funkcja wywołuje funkcję free_pixels().

4. **Funkcja free_pixels()**

   Poprzednio:
   Brak

   Aktualnie:
   void free_pixels(void);

   Opis:
   Funkcja free_pixels() odpowiada za zwalnianie pamięci używanej podczas tworzenia obrazów PNG. Jest funkcją nieprzyjmującą żadnych argumentów.

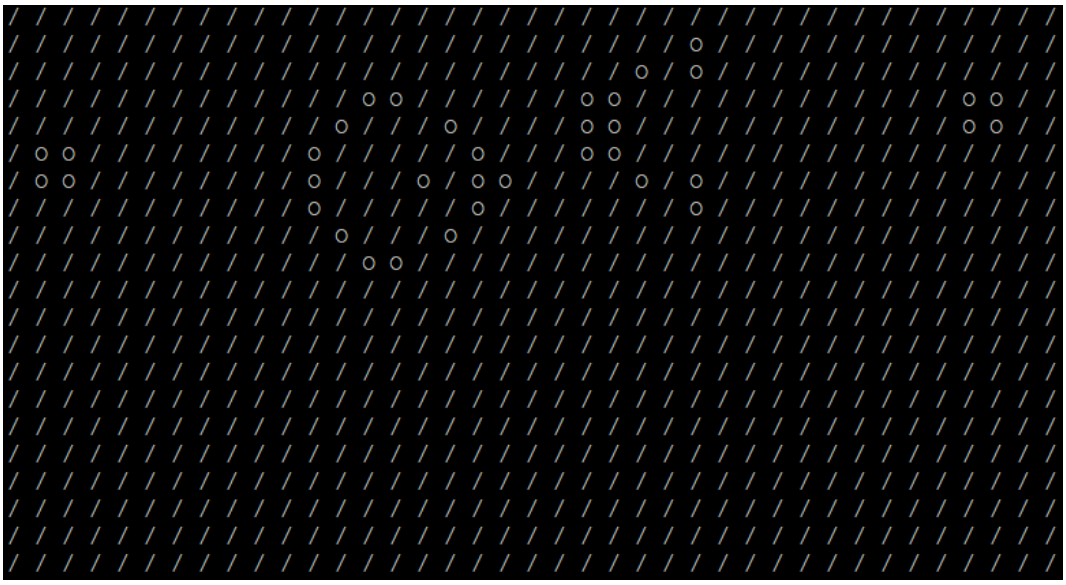
4 Prezentacja działania

4.1 Przykładowe wywołanie

Poniżej przedstawione jest przykładowe wywołanie programu z następującymi argumentami ./a.out -n 3 -i in.txt -a O -d / -w 2, gdzie:

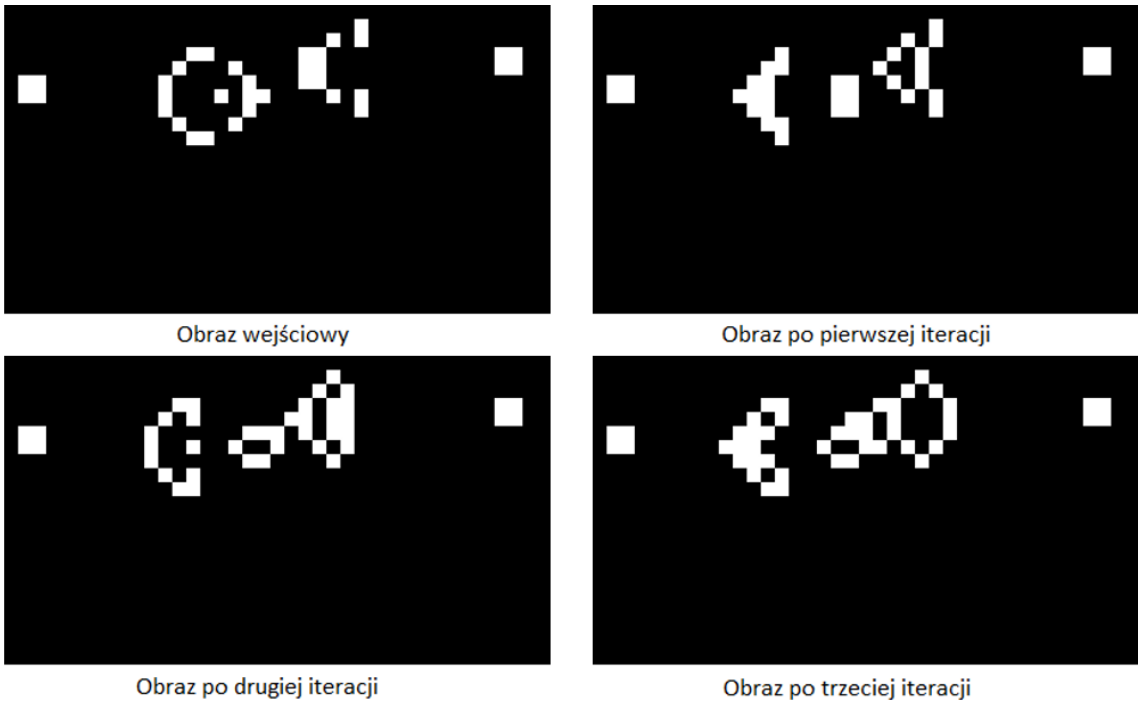
1. -n 3: liczba generacji ma wynosić 3
2. -i in.txt: plik wejściowy ma nazwę in.txt
3. -a O: żywe komórki są oznaczone znakiem 'O'
4. -d /: martwe komórki są oznaczone znakiem '/'
5. -w 2: program wygeneruje obrazy PNG

4.2 Plik wejściowy



Rysunek 2: Plik tekstowy in.txt

4.3 Wynik wywołania



Rysunek 3: Obraz wejściowy oraz jego 3 kolejne iteracje

5 Podsumowanie testów modułów

W ramach programu wykonaliśmy niezbędne testy, dzięki którym mogliśmy obserwować zachowania programu po wprowadzeniu różnych argumentów. Do każdego modułu zostały wykonane niezależne testy. Dodatkowo został wykonany test sprawdzający wyciek pamięci.

Moduł ReadFile:

- test_read_dimension.c
- test_fill_in_grid.c
- file_test_read_file.txt

Moduł Game:

- test_is_alive.c

Moduł WriteFile:

- test_print_to_screen.c
- test_fill_in_file.c
- file_test_write_file.txt

Moduł PNG:

- test_png.c
- Obraz1_TEST.png

Pamięć:

- Valgrind

Poniżej zostaną zaprezentowane testy, po jednym dla każdego modułu.

5.1 Moduł ReadFile - test_fill_in_grid.c

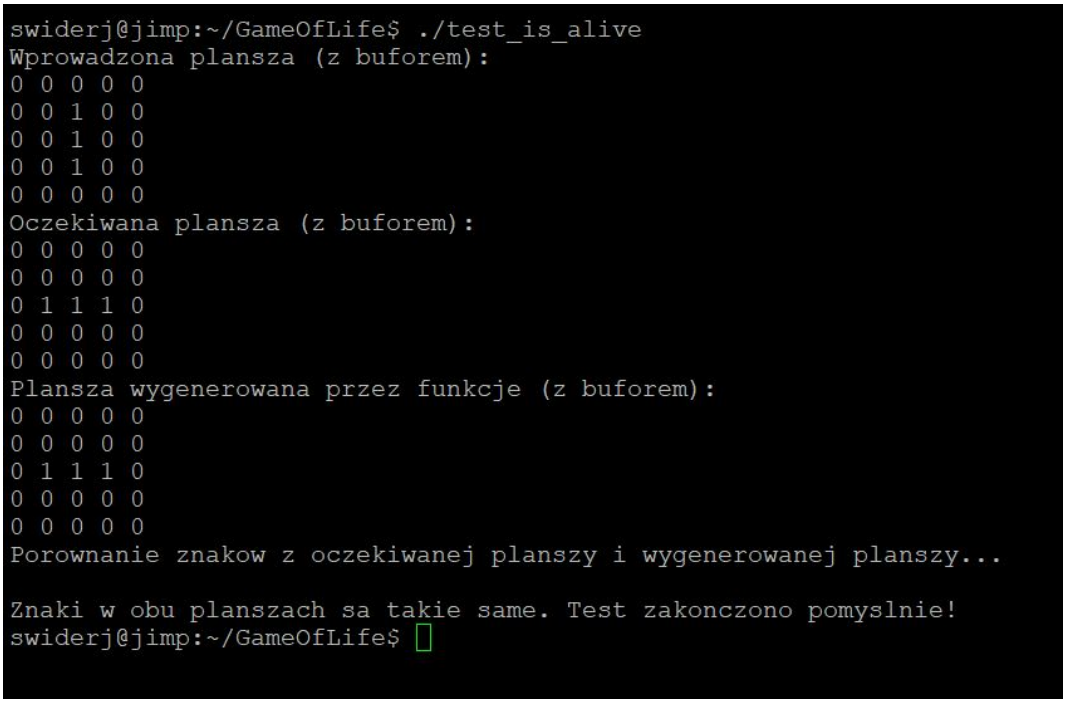
Test funckji fill_in_grid sprawdza czy funkcja poprawnie zapisuje do struktury pierwszą planszę z pliku wejściowego. Test do działania potrzebuje danego pliku wejściowego z opisaną planszą. W pogramie zadeklarowana jest plansza identyzna jak w pliku wejściowym. Program wywołuje funkcję fill_in_grid, która jako jeden z argumentów przyjmuje nazwę pliku wejsciowego. Na jego podstawie funkcja wypełnia planszę w strukturze grid_t (czyli w takiej formie w jakiej plansza będzie funkcjonować w gotowym projekcie). Program następnie sprawdza czy wszystkie znaki w obu planszach (tej zadeklarowanej i tej wypełnionej przez funkcję) są identyczne i wypisuje stosowny komunikat.

```
skibap1@jimp:~/GameOfLife$ gcc ./test_fill_in_grid.c
skibap1@jimp:~/GameOfLife$ ./a.out
Plansza zadeklarowana w programie:
0 0 0 0 0 0
0 1 0 1 0 0
0 1 0 1 0 0
0 0 0 0 0 0
Plansza wypelniona przez funkcje:
0 0 0 0 0 0
0 1 0 1 0 0
0 1 0 1 0 0
0 0 0 0 0 0
Przy porownaniu 2 plansz (jedna zapelniona poprzez funkcje,
druga przygotowana do porownania) otrzymalismy 8 elementow
takich samych na tych samych pozycjach w macierzy 2 x 4.
Test zakonczony pomyslnie!
```

Rysunek 4: Test funckji fill_in_grid() z modułu ReadFile

5.2 Moduł Game - test_is_alive.c

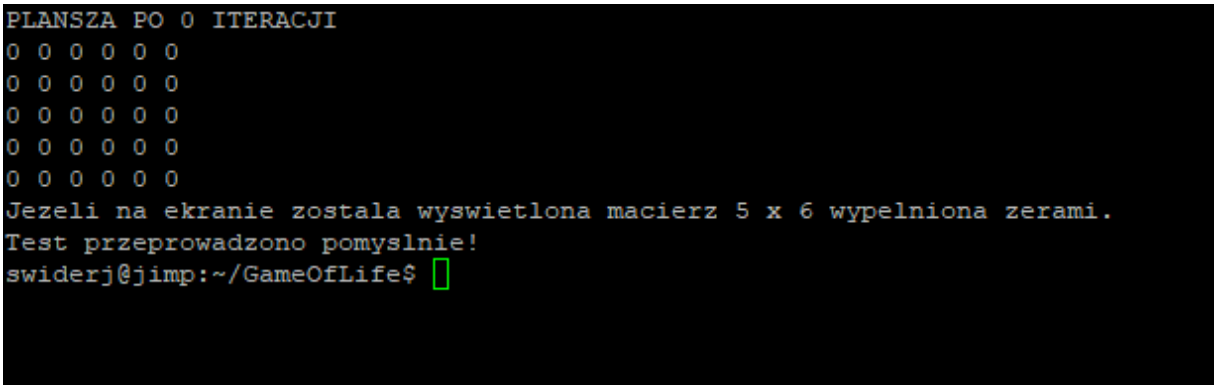
Test funkcji is_alive() polega na przekazaniu do funkcji matrycy dwuwymiarowej z wybranym układem komórek, który po przeprowadzeniu jednej iteracji gry w życie Conway'a wyprodukuje znaną dla nas wcześniej planszę (w naszym przykładzie jest to oscylator "blinker"). Program wywołuje raz funkcję is_alive() dla wybranej planszy, zapisuje wygenerowaną planszę, a następnie porównuje jej znaki ze znakami w oczekiwanej planszy wcześniej zdefiniowanej w programie. W zależności od tego czy test się powiódł program wypisze stosowny komunikat



Rysunek 5: Test funckji is_alive() z modułu Game

5.3 Moduł WriteFile - test_print_to_screen.c

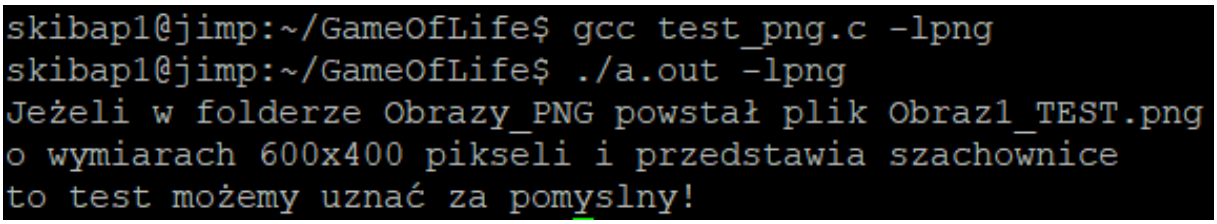
Test funkcji print_to_screen() ma sprawdzić czy poprawnie wypisuje ona planszę na wyjście standardowe. Program wpierw tworzy strukturę z planszą i alokuje dla niej pamięć na podstawie zdefiniowanych na początku liczby wierszy i kolumn (odpowiednio #define ROWS, #define COLUMNS). Jeżeli plansza wypisana przez funkcję będzie zgadzać się z opisem tego jak powinna wyglądać, to znaczy że test został przeprowadzony pomyślnie.



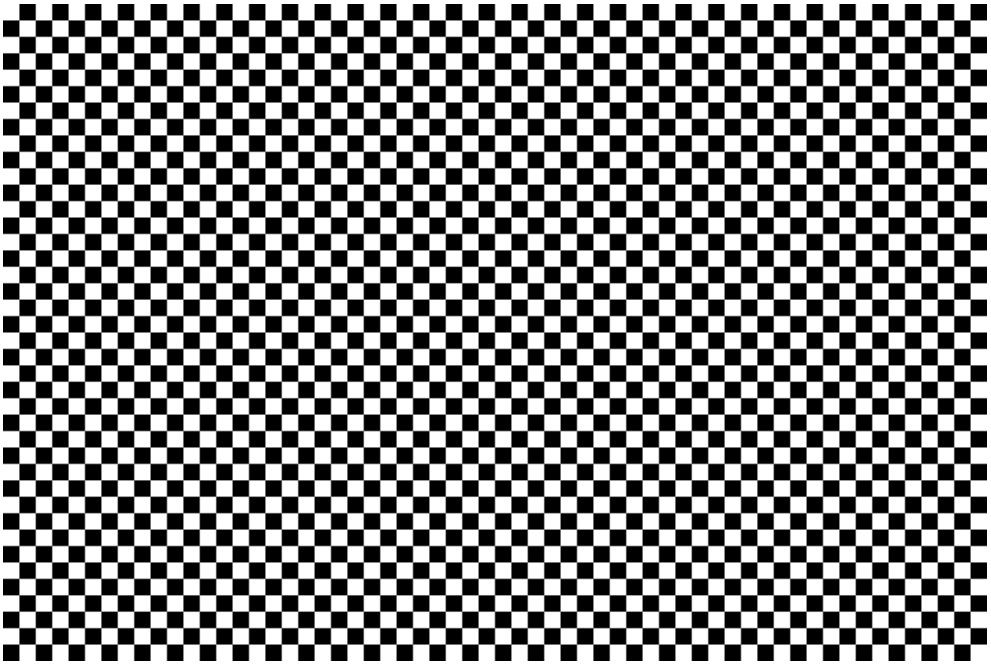
Rysunek 6: Test funckji print_to_screen() z modułu WriteFile

5.4 Moduł PNG - test_png.c

Test funkcji test_png.c ma za zadanie utworzyć obraz o wymiarach 600 x 400 pikseli. Obraz ma być wypełniony kwadratami 10 x 10 pikseli oraz powinien przedstawiać czano-białą szachownicę. Obraz ten powinien zostać zamieszczony w folderze Obrazy_PNG. Jeżeli obraz zostanie wygenerowany zgodnie z oczekiwaniami, możemy uznać test za udany.



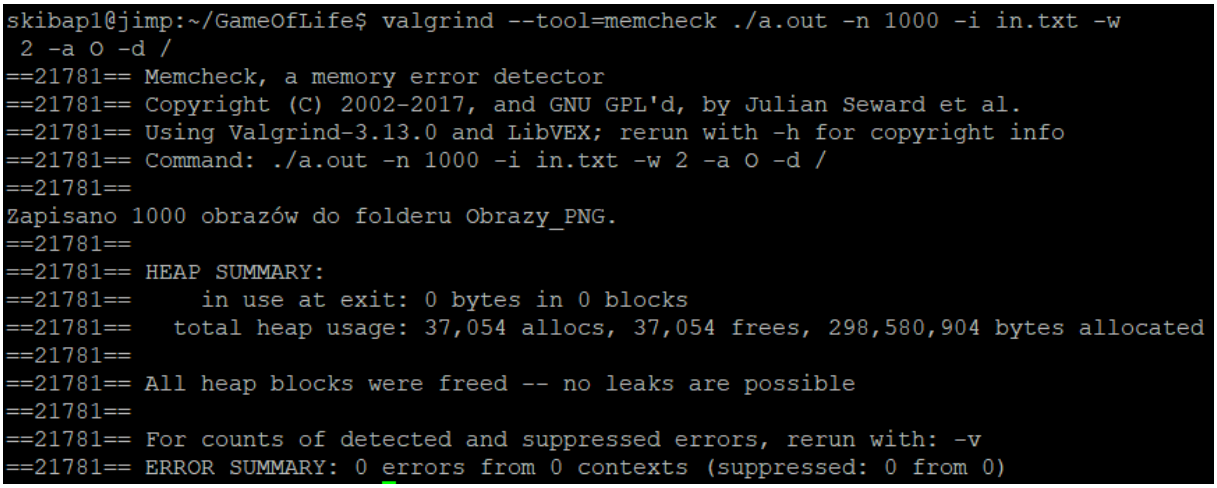
Rysunek 7: Test funkcji png() z modułu PNG



Rysunek 8: Wygenerowany obraz przez program test_png.c

5.5 Pamięć - valgrind

Narzędzie valgrind służy do sprawdzenia, czy alokowana pamięć jest zwalniana. Valgrind został uruchomiony w konfiguracji z progremem, który zapisywał iteracje do pliku PNG oraz wykonał ich tysiąc. Program w tym celu zaalokował 298 580 904 bajtów i dokładnie tyle samo zwolnił. Potwierdza to, że w programie nie ma wycieków pamięci, co można uznać za test zakończony powodzeniem.



Rysunek 9: Sprawdzenie wycieków pamięci za pomcą narzędzia valgrind