

Krzysztof Dąbrowski i Jakub Bogusz

Raport końcowy – Gra w życie

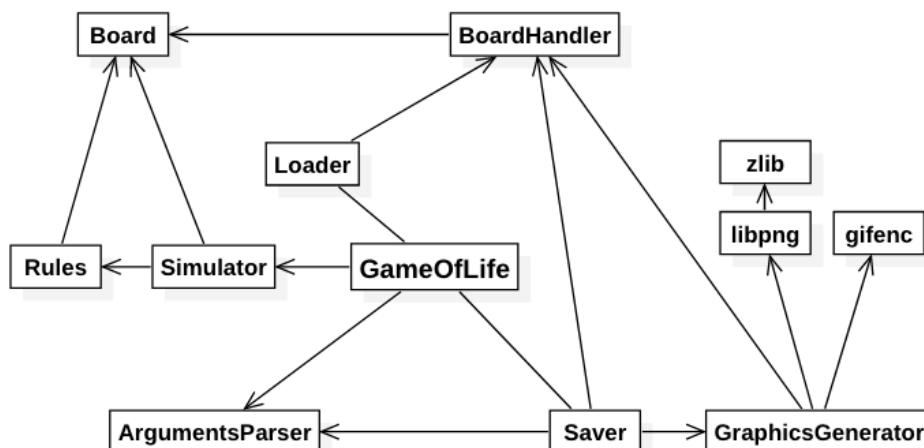
12 kwietnia 2019

Spis treści

1. Ostateczny projekt modułów	1
2. Opis modyfikacji	2
2.1. GameOfLife	2
2.2. ArgumentsParser	2
2.3. Simulator	2
2.4. BoardSaver	2
2.5. GraphicsGenerator	3
Nowe funkcje	3
3. Prezentacja działania	3
3.1. Generacja obrazów z wczytaniem stanu z pliku	3
Powstałe obrazy	4
3.2. Generacja plików tekstowych z wczytaniem losową planszą początkową	5
3.3. Generacja plików gif z losową planszą	5
3.4. Wyświetlenie kolejnych pokoleń w konsoli z planszą pobieraną z pliku	5
4. Podsumowanie testów modułów	6
4.1. Wygenerowany raport	6
4.2. Wnioski	6
5. Analiza pamięci	6

1. Ostateczny projekt modułów

Zmodyfikowany diagram modułów:



2. Opis modyfikacji

W trakcie pracy nad projektem okazało się, że niektóre problemy można rozwiązać lepiej niż przewiduje to specyfikacja implementacyjna. Pojawiły się również nieprzewidziane problemy z wyświetlaniem bardzo małych obrazów przez typowe programy do prezentacji grafik. Z tych powodów wprowadzone zostały pewne zmiany.

2.1. GameOfLife

Specyfikacja nie przewidywała funkcji odpowiedzialnej za główny przebieg sterowania w programie. Dodano więc funkcję `void runProgram(int argc, char **args)`. Koordynuje ona pracę wszystkich pozostałych modułów.

2.2. ArgumentsParser

Do typu wyliczeniowego `FileType` (dawniej `Type`) została dodana nowa wartość „OUT” reprezentująca wypisywanie kolejnych stanów na standardowe wyjście.

Do struktury `Config` zostały dodane nowe pola `sizeX` i `sizeY` reprezentujące wymiary planszy.

Dodano funkcję `void disposeConfig(Config *config)`, która zwalnia pamięć zarezerwowaną na tę strukturę.

2.3. Simulator

Zmieniono funkcję `Board** simulate(Board* b, Config* p)` tak aby na podstawie planszy początkowej generowała tablicę wskaźników na struktury będące reprezentacjami kolejnych pokoleń - historię stanów.

Dodano funkcję `int calcHistorySize(Config* c)` zwracającą ilość generowanych stanów – długość historii stanów.

Dodano funkcję `Board** stepSimulate(Board** b, Config* p)` przerzucającą historię stanów tak by zawierała tylko co n-ty stan, gdzie n to wartość flagi „step”.

2.4. BoardSaver

Nazwa modułu została zmieniona na `Saver`, a jego struktura gruntownie zmodyfikowana. Aktualnie kontroluje wszystkie formy zapisu i wyświetlenia wyniku.

Aktualna postać modułu:

`void saveAsPng(Board** history, Config* config, int i)` – Zapis kolejnych pokoleń do plików png.

`Board** history` – tablica wskaźników na struktury reprezentujące kolejne pokolenia,
`Config* config` – wskaźnik na strukturę ustawień generacji kolejnych pokoleń,
`int i` – numer aktualnie generowanego pokolenia. Potrzebny do generowania nazw plików.

`void saveAsGif(Board** history, Config* config, int historySize)` – Zapis kolejnych pokoleń do pliku gif.

`Board** history` – tablica wskaźników na struktury reprezentujące kolejne pokolenia,
`Config* config` – wskaźnik na strukturę ustawień generacji kolejnych pokoleń,
`int historySize` – Liczba pokoleń w historii

`void saveAsTxt(Board** history, Config* config, int i)` – Zapis kolejnych pokoleń do plików txt.

`Board** history` – tablica wskaźników na struktury reprezentujące kolejne pokolenia,
`Config* config` – wskaźnik na strukturę ustawień generacji kolejnych pokoleń,
`int i` – numer aktualnie generowanego pokolenia. Potrzebny do generowania nazw plików.

`void printToStdout(Board** history, Config* config, int i)` – Wyświetlenie kolejnych pokoleń w terminalu.

`Board** history` – tablica wskaźników na struktury reprezentujące kolejne pokolenia,
`Config* config` – wskaźnik na strukturę ustawień generacji kolejnych pokoleń,
`int i` – numer aktualnie generowanego pokolenia. Potrzebny do generowania nazw plików.

2.5. GraphicsGenerator

W tym module zostały dodane nowe funkcje pomocnicze związane z przeskalowywaniem obrazu. Okazało się, że po stworzeniu obrazu o rozmiarze w pikselach takich jak plansza, domyślnie jest on wyświetlany w bardzo małej wielkości przez typowe programy. By tego uniknąć dodana została funkcjonalność przeskalowywania obrazów do minimalnej wielkości.

By skalowanie było uniwersalne pracuje ono na wartościach pikseli, z których ma powstać grafika. Dzięki temu jest niezależne od palet barw, które są różne dla generowania png i gif. Do przechowywania wartości pikseli używany jest nowy typ `Pixel`.

Skalowanie może tylko zwiększyć wymiary obrazu.

2.5.1. Nowe funkcje

Na potrzeby skalowania zostały napisane następujące funkcje:

`Pixel *translateBoardToPixels(Board *board, Pixel valueOfAlive, Pixel valueOfDead)` – Zamiana planszy na tablicę pikseli.

`Board *board` – wskaźnik na planszę, która ma być przedstawiona jako piksele,
`Pixel valueOfAlive` – wartość pikseli dla żywych komórek,
`Pixel valueOfDead` – wartość pikseli dla martwych komórek.

Zwracany jest wskaźnik na dynamicznie przydzieloną tablicę pikseli odpowiadającą otrzymanej planszy.

`void getUpscaledImageSize(int originalX, int originalY, int *newX, int *newY)` – Obliczenie wymiarów obrazu po przeskalowaniu.

`int originalX` – szerokość oryginalnego obrazu,
`int originalY` – wysokość oryginalnego obrazu,
`int *newX` – wskaźnik na zmienną typu int, gdzie zostanie zapisana nowa szerokość planszy,
`int *newY` – wskaźnik na zmienną typu int, gdzie zostanie zapisana nowa wysokość planszy.

`Pixel *upscaleImage(const Pixel *original, int imageX, int imageY)` – Przeskalowanie obrazu.

`const Pixel *original` – wskaźnik na tablicę pikseli reprezentującą oryginalny obraz,
`int imageX` – szerokość oryginalnego obrazu,
`int imageY` – wysokość oryginalnego obrazu,

Zwracany jest wskaźnik na dynamicznie przydzieloną tablicę pikseli odpowiadającą przeskalowanemu obrazowi.

2.5.2. Zmiany już istniejących funkcji

Funkcja `savePng` nie przyjmuje ustawień `Config*` c. Nie są one potrzebne do generowania obrazu. Zamiast tego przyjmuję ścieżkę do miejsca, gdzie będzie wygenerowany. Dodatkowo zamiast tekstu `char* data` jako pierwszy argument przyjmuje strukturę planszy. Dzięki temu została wyeliminowana zbędna serializacja i parsowanie planszy.

Nazwa funkcji `saveGif` została zmieniona na `saveHistoryAsGif` ponieważ w bibliotece do tworzenia gifów istnieć już funkcja o takiej nazwie. Funkcja ta również zamiast struktury ustawień otrzymuje ścieżkę do miejsca, gdzie plik ma zostać wygenerowany.

3. Prezentacja działania

Przykłady wywołania programu z różnymi flagami.

3.1. Generacja obrazów z wczytaniem stanu z pliku

Wywołanie programu:

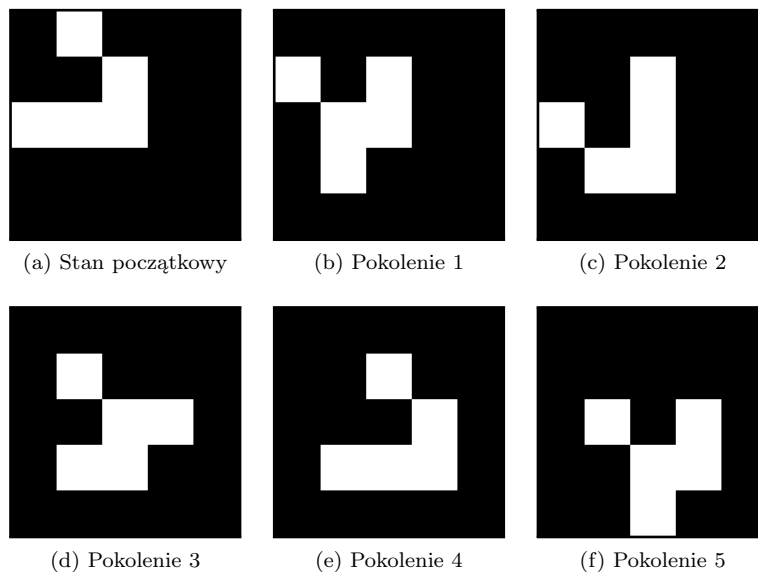
```
./game-of-life -f input.txt -t png -o raport -n 5
```

Argumenty:

- `-f input.txt` – Stan początkowy jest wczytany z pliku „input.txt”,
- `-t png` – Zostanie wygenerowana seria plików .png z kolejnymi pokoleniami,
- `-o raport` – Wygenerowane pliki zostaną zapisane do podfolderu w folderze „raport”,
- `-n 5` – Zostanie wygenerowanych 5 następnych pokoleń .png (ze stanem początkowym 6 obrazków).

Powstałe obrazy

Program wygenerował poniższe pliki. Przedstawiają ruch struktury szybowca.



Rysunek 1. Wygenerowane pliki

3.2. Generacja plików tekstowych z wczytaniem losową planszą początkową

Wywołanie programu:

```
./game-of-life -s 5x5 -t txt -o raport -n 10
```

Argumenty:

- **-s 5x5** – Zostanie wygenerowana losowa plansza początkowa w rozmiarach 5 na 5 komórek,
- **-t txt** – Zostanie wygenerowana seria plików .txt z kolejnymi pokoleniami,
- **-o raport** – Wygenerowane pliki zostaną zapisane do podfolderu w folderze „raport”,
- **-n 3** – Zostanie wygenerowanych 3 następnych pokoleń .png (ze stanem początkowym 4 plików tekstowych).

Wygenerował pliki o poniższej zawartości:

```
5 5
1 1 1 1 1
0 0 1 1 0
1 0 1 1 1
1 1 0 1 0
1 1 0 0 1
```

Jako pokolenie początkowe.

```
5 5
0 0 0 0 0
1 1 0 0 0
0 0 0 0 0
1 0 0 0 0
0 1 0 0 0
```

Jako pokolenie drugie.

```
5 5
0 1 0 0 1
1 0 0 0 0
1 0 0 0 1
0 0 0 0 0
1 1 1 0 0
```

Jako pokolenie pierwsze.

```
5 5
0 0 0 0 0
0 0 0 0 0
1 1 0 0 0
0 0 0 0 0
0 0 0 0 0
```

Jako pokolenie trzecie.

3.3. Generacja plików gif z losową planszą

Wywołanie programu:

```
./game-of-life -s 20x20 -o raport -n 50
```

Argumenty:

- **-s 20x20** – Zostanie wygenerowana losowa plansza początkowa w rozmiarach 20 na 20 komórek,
- **-o raport** – Wygenerowany plik zostanie zapisany z folderze raport,
- **-n 50** – Zostanie wygenerowanych 50 następnych pokoleń w formie jednego pliku .gif (ze stanem początkowym 51 klatek).

Nie podanie formatu pliku wynikowego spowoduje wygenerowanie pliku .gif. Wyniki pracy programu dostępne [pod linkiem](#).

3.4. Wyświetlenie kolejnych pokoleń w konsoli z planszą pobieraną z pliku

Wywołanie programu:

```
./game-of-life -f input.txt -t out -n 20 -d 500
```

Argumenty:

- **-f input.txt** – Stan początkowy jest wczytany z pliku „input.txt”,
- **-t out** – kolejne pokolenia zostaną wyświetlone na konsoli,

- `-n 50` – Zostanie wygenerowanych 50 następnych pokoleń w formie jednego pliku `.gif` (ze stanem początkowym 51 klatek),
- `-d 500` – Kolejne pokolenia będą wyświetlane co 500ms.

Takie wywołanie programu wyświetli kolejne pokolenia w terminalu zachowując pół sekundy odstępy między kolejnymi stanami planszy w formacie podobnym do wyników z plików tekstowych (bez wymiarów planszy).

4. Podsumowanie testów modułów

Do pisania oraz automatyzacji wykonania testów jednostkowych został wykorzystany framework *CUnit*. Dzięki temu na dowolnym etapie projektu można było łatwo sprawdzić poprawność już istniejącego kodu.

4.1. Wygenerowany raport

Wyniki testów jednostkowych przedstawia poniższy raport.

```
CUnit - A unit testing framework for C - Version 2.1-3
http://cunit.sourceforge.net/

Suite: Board tests
  Test: To string test ...passed
Suite: Loader tests
  Test: Parse small file test ...passed
Suite: Rules tests
  Test: Dead cell stays dead ...passed
  Test: Dead cell comes to live ...passed
  Test: Alive cell dies from overpopulation ...passed
  Test: Alive cell dies from loneliness ...passed
  Test: Alive cell stays alive ...passed
Suite: Simulator tests
  Test: Simulate one next generation on small board ...passed
  Test: Simulate ten generations on not so small board, but not a big one too ...passed

Run Summary:      Type  Total    Ran Passed Failed Inactive
                  suites    4      4   n/a    0      0
                  tests    9      9    9     0     0
                  asserts  41     41   41     0    n/a

Elapsed time =    0.001 seconds
```

4.2. Wnioski

Z raportu testów jednostkowych wynika, że wszystkie wszystkie sprawdzone funkcjonalności dają prawidłowe wyniki. Wszystkie testy są oznaczone jako `...passed` oraz liczba zapewnień, których wyniki są poprawne jest równa liczbie wszystkich zapewnień.

4.3. Testy generacji wyników

Moduły zapisujące i przedstawiające wyniki symulacji zostały przetestowane poprzez ręczne testy integracyjne.

5. Analiza pamięci

Raport programu `valgrind` zwraca następujące wyniki:

```
==18728== LEAK SUMMARY:
==18728== definitely lost: 0 bytes in 0 blocks
==18728== indirectly lost: 0 bytes in 0 blocks
==18728== possibly lost: 72 bytes in 3 blocks
==18728== still reachable: 23,728 bytes in 28 blocks
==18728== suppressed: 18,077 bytes in 153 blocks

==18728== ERROR SUMMARY: 0 errors from 0 contexts
```

2 pierwsze linie raportu informują o braku definitywnych wycieków pamięci, co oznacza że program prawidłowo zarządza pamięcią.

Wycieki oznaczone jako `still reachable` oraz `possibly lost` spowodowane są przez funkcje bibliotek `ctime` oraz `libpng`, więc nie mamy na nie wpływu i nie jesteśmy w stanie ich wyeliminować. Załączamy fragmentu raportu opisujące te wycieki:

possible leaks:

```
==18752== 72 bytes in 3 blocks are possibly lost in loss record 34 of 60
==18752== at 0x1000B26EA: calloc (in /usr/local/Cellar/valgrind/3.14.0/lib/valgrind/vgpreload_memcheck-amd64-darwin.so)
==18752== by 0x1007AD7C2: map_images_nolock (in /usr/lib/libobjc.A.dylib)
==18752== by 0x1007C04E0: map_images (in /usr/lib/libobjc.A.dylib)
==18752== by 0x1000DC64: dyld::notifyBatchPartial(dyld_image_states, bool, char const* (*)(dyld_image_states, unsigned int, dyld_image_info const*), bool, bool) (in /usr/lib/dyld)
==18752== by 0x1000DE39: dyld::registerObjCNotifiers(void (*)(unsigned int, char const* const*, mach_header const* const*), void (*)(char const*, mach_header const*), void (*)(char const*, mach_header const*)) (in /usr/lib/dyld)
==18752== by 0x10027871D: _dyld_objc_notify_register (in /usr/lib/system/libdyld.dylib)
==18752== by 0x1007AD073: _objc_init (in /usr/lib/libobjc.A.dylib)
==18752== by 0x100202B34: _os_object_init (in /usr/lib/system/libdispatch.dylib)
==18752== by 0x100202B1B: libdispatch_init (in /usr/lib/system/libdispatch.dylib)
==18752== by 0x1001119C2: libSystem_initializer (in /usr/lib/libSystem.B.dylib)
==18752== by 0x10001FAC5: ImageLoaderMach0::doModInitFunctions(ImageLoader::LinkContext const&) (in /usr/lib/dyld)
==18752== by 0x10001FCF5: ImageLoaderMach0::doInitialization(ImageLoader::LinkContext const&) (in /usr/lib/dyld)
```

still reachable:

```
==18805== 18,280 bytes in 1 blocks are still reachable in loss record 60 of 60
==18805== at 0x1000B26EA: calloc (in /usr/local/Cellar/valgrind/3.14.0/lib/valgrind/vgpreload_memcheck-amd64-darwin.so)
==18805== by 0x100351930: tzsetwall.basic (in /usr/lib/system/libsystem.c.dylib)
==18805== by 0x1003537C9: localtime (in /usr/lib/system/libsystem.c.dylib)
==18805== by 0x100353945: ctime (in /usr/lib/system/libsystem.c.dylib)
==18805== by 0x1000039AF: setup (Saver.c:5)
==18805== by 0x100003A2E: saveCommon (Saver.c:12)
==18805== by 0x100003E59: saveAsPng (Saver.c:53)
==18805== by 0x10000405E: runProgram (main.c:83)
==18805== by 0x100003F31: main (main.c:32)
```