

Raport końcowy – Gra w życie

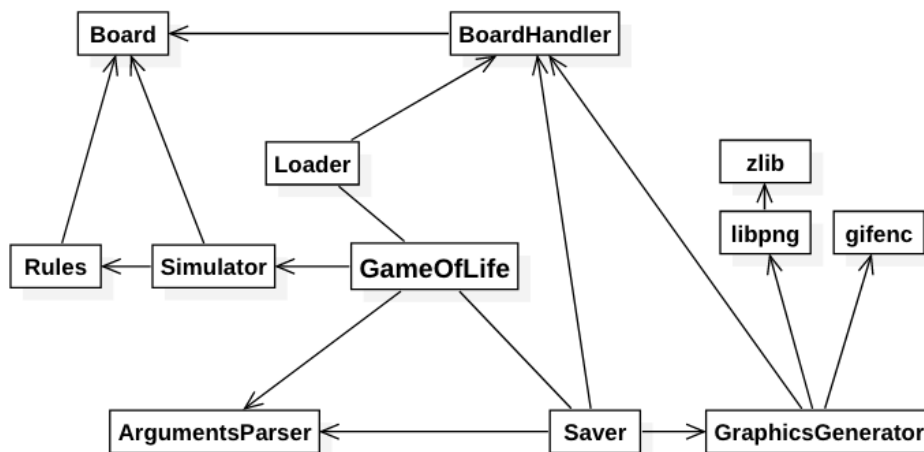
9 kwietnia 2019

Spis treści

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

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 przewidywał funkcji odpowiedzialnej z 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.

3. Prezentacja działania

Przykłady wywołania programu z różnymi flagami.

3.1. Generacja obrazów z wczytaniem stanu z pliku

Wywołanie programu:

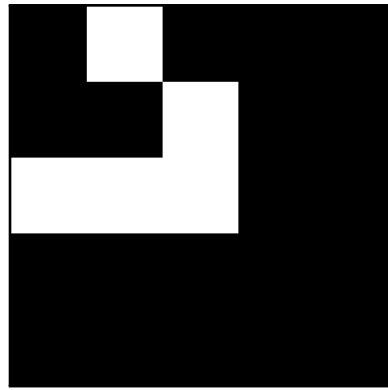
```
./game-of-life -f input.txt -t png -o raport -n 6
```

Argumenty:

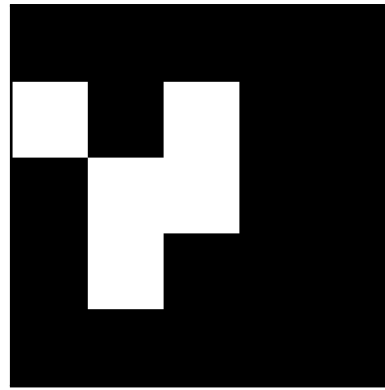
- `-f input.txt` – Stan początkowy jest wczytany z pliku „input.txt”,
- `-t png` – Zostanie wygenerowana seria plików .png z kolejnymi pokoleniami,
- `-o raport` – Wygenerowane pliki zostaną zapisane do podfolderu w folderze „raport”,
- `-n 6` – Zostanie wygenerowanych 6 następnych pokoleń .png (ze stanem początkowym 7 obrazków).

Powstałe obrazy

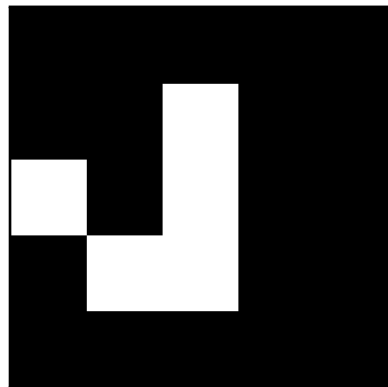
Program wygenerował poniższe pliki. Przedstawiają ruch struktury szybowca.



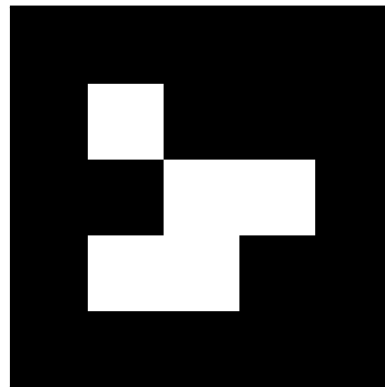
(a) Pokolenie 0



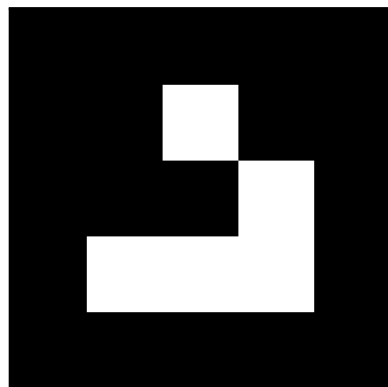
(b) Pokolenie 1



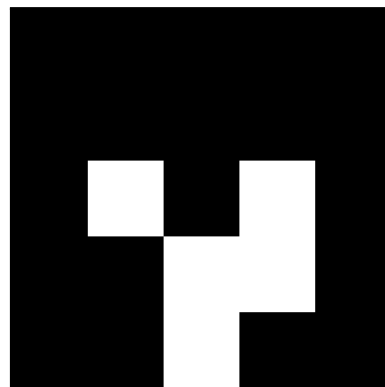
(c) Pokolenie 2



(d) Pokolenie 3



(e) Pokolenie 4



(f) Pokolenie 5

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 sekundowe 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
```

Run Summary:	Type	Total	Ran	Passed	Failed	Inactive
	suites	4	4	n/a	0	0
	tests	8	8	8	0	0
	asserts	27	27	27	0	n/a

Elapsed time = 0.001 seconds

4.2. Wnioski

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

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)

```