Specyfikacja implementacyjna projektu *Gra w Życie* w języku C

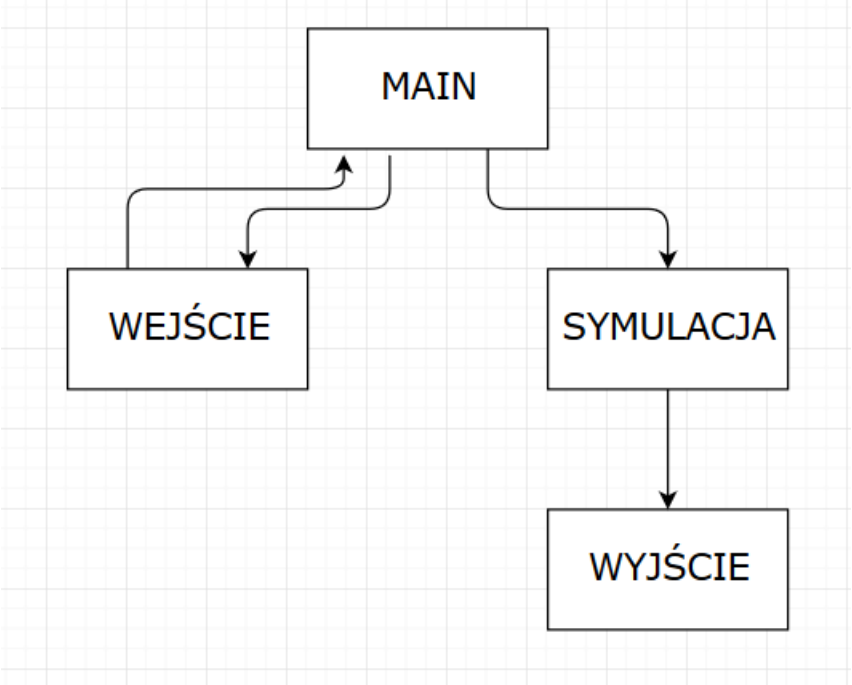
Chabik Jan (291060), Łuczak Mateusz (291088)

15 marca 2018

Spis treści

1 Diagram modułów

1.1 Diagram



1.2 Opis diagramu

- `MAIN` - Główna funkcja programu. Przekazuje dane do funkcji sprawdzającej wejście, tworzy macierze dla mapy automatu komórkowego i wysyła parametry wejścia do modułu `SYMULACJA`;
- `WEJŚCIE` - Funkcja sprawdzająca parametry wejścia, wyświetla odpowiednie komunikaty o błędach i wysyła informacje do funkcji głównej;
- `SYMULACJA` - Funkcja odpowiadająca za generowanie nowych map automatu komórkowego na podstawie poprzedników i reguł sąsiedztwa. Odpowiednie dane (mapa, numer generacji, katalog wyjściowy) są wysyłane do modułu `WYJŚCIE`;
- `WYJŚCIE` - Funkcja odpowiadająca za wypisanie danych wynikowych w formatach `.txt` i `.png` do przygotowanego katalogu. Na życzenie użytkownika, wypisuje również informacje do terminala.

2 Funkcje

2.1 Funkcje modułu głównego

- `INIT_MATRIX` - funkcja tworząca macierze, które będą przechowywać dane dotyczące mapy automatu komórkowego. Zostaną one utworzone natychmiastowo po sprawdzeniu danych wejściowych i wczytaniu wymiarów mapy automatu.

2.2 Funkcje modułu wejścia

- `CHECK` - funkcja interpretująca informacje dotyczące danych wejściowych:
  - `CHECK_INPUT_FILE` - funkcja sprawdzająca plik wejściowy `.txt` pod względem błędów odczytu lub błędów pliku;
  - `CHECK_OUTPUT_CATALOG` - funkcja sprawdzająca katalog wyjściowy pod względem błędów zapisu (ewentualny brak dostępu do katalogu);
  - `CHECK_ARGS` - funkcja sprawdzająca argumenty podane przez użytkownika podczas uruchamiania programu.

2.3 Funkcje modułu symulacji

- `MAINF` - Funkcja wywołująca wybraną ścieżkę na podstawie wybranego sąsiedztwa i opcji dialogowej. Po zakończeniu generowania nowej mapy, jest ona kopiowana w miejsce jej poprzednika (do pierwszej macierzy);
- `SASIEDZTWOM` - Funkcja obsługująca sąsiedztwo Moore'a;
- `SASIEDZTWON` - Funkcja obsługująca sąsiedztwo von Neumanna.

2.4 Funkcje modułu wyjścia

- `OUTPUT_TO_TERM` - Funkcja, która wypisuje dane do terminala i obsługuje dialog z użytkownikiem, który ma wybór pomiędzy zapisem aktualnej generacji do pliku `.png`, lub jego pominięciem. Zapis do pliku `.txt` jest automatyczny i użytkownik nie może wybrać, czy generacja ma zostać pominięta. Dzięki temu, zapisane zostają wszystkie generacje i użytkownik może wywołać program dla wybranego pliku `.txt`. W innym wypadku, istniałoby ryzyko, gdzie nie zostałyby zapisane żadne pliki w formacie tekstowym oraz niemożliwe byłoby przywrócenie wybranej generacji bez tworzenia ich raz jeszcze;
- `MAIN_OUTPUT` - Funkcja zapisująca dane do pliku `.txt` oraz `.png`.

3 Metodyka wersjonowania

- Wersjonowanie oprogramowania następuje na systemie kontroli wersji Git;
- Każda z części kodu jest udostępniana na nowych gałęziach (*branch*);
- Wersje dokumentów obarczone będą nazwą z dopiskiem `beta` lub `FINAL`, oraz numerem wersji:
  - `beta` - robocza wersja dokumentu;
  - `FINAL` - ostateczna wersja dokumentu;
  - numer wersji - numer oznaczający aktualną kompilację. Określony zostaje w formacie X.Y, gdzie X to numer kompilacji, a Y numer poprawki. Y jest opcjonalne i występuje tylko przy małych zmianach w dokumencie (takich jak przykładowo poprawa literówki).
- Wersje dokumentów obarczone będą nazwą z dopiskiem `beta` z numerem kompilacji w razie wysłania do repozytorium wersji roboczej, `FINAL` w przypadku wersji ostatecznej. W przypadku dodatkowych poprawek, będzie posiadać numer kompilacji.

4 Kompilator i wersja języka

4.1 Kompilator

Używanym kompilatorem będzie `gcc`. Dzięki niemu możliwe jest dobranie wersji języka C. Jest to również kompilator ogólnodostępny i darmowy.

4.2 Wersja języka

Wybraną wersją języka C jest `C99` ze względu na dodatkowe udogodnienia przy pisaniu kodu. Jednym z przykładów może być możliwość deklarowania zmiennej podczas tworzenia pętli `for`.
Przykład:

```
for(int i=0;i<10;i++) /* kod */
```

Do kompilacji kodu w tym standardzie konieczne jest użycie flagi `-std=c99`.
Przykład:

```
gcc -std=c99 foo.c
```

5 Użyte biblioteki

- `png.h` - obsługa obrazów w formacie `.png`. Do użycia konieczne jest doinstalowanie biblioteki `libpng`, która jest dostępna do pobrania pod adresem: `https://libpng.sourceforge.io/index.html`

6 Testowanie i konwencja

6.1 Testowanie

Testy zostaną wykonane przez nas ręcznie. Testować będziemy każdą funkcję z osobna, a potem każdy moduł.

6.2 Konwencja

Będziemy testować program przez napisanie funkcji przypisanych do poszczególnych funkcjonalności. Zamiast jednej funkcji sprawdzającej działanie modułu wejścia, napiszemy takich funkcji wiele i każda z nich będzie testować jedną, unikatową funkcjonalność (np. poprawny komunikat przy problemie z plikiem wejścia, funkcja będzie się nazywać `shouldShowInputFileError()`). Robimy tak, by jak najszybciej identyfikować błędy w naszym kodzie.

6.3 Kontrola wycieków pamięci

Do kontroli wycieków pamięci będziemy używać programu `Valgrind`. Jest to program łatwy w obsłudze, skuteczny i darmowy. Pozwala sprawdzić, w którym miejscu następuje wyciek, oraz ile danych zostaje utraconych.