Laboratorium 11: „Obsługa terminala - biblioteka curses”

mgr inż. Leszek Ciopiński
dr inż. Arkadiusz Chrobot

4 września 2026

1. Wprowadzenie

W systemach serwerowych, graficzny interfejs użytkownika (GUI) jest rzadko wykorzystywany, gdyż z reguły dostęp odbywa się zdalnie. W związku z tym może być wyłączony, żeby zaoszczędzić moc obliczeniową maszyny. Praca w konsoli, pomimo licznych zalet, jak możliwość pisania skryptów, w niektórych sytuacjach może być mniej wygodna. Aby jednak nie było potrzeby uruchamiania GUI, można uruchomić aplikację konsolową, która imituje okna. Bezpośrednie manipulowanie pozycją kursora na ekranie przez programistę jest jednak niewygodne. Aby ułatwić to zadanie, możliwe jest użycie biblioteki do organizowania układu okien w konsoli. Jedną z takich bibliotek to curses, opisana w niniejszej instrukcji.

2. Instalacja biblioteki

Instalacja biblioteki zależy od systemu operacyjnego.

2.1. Systemy kompatybilne z Uniksem

Oryginalna biblioteka przygotowana jest na środowiska wywodzące się z systemów Unix. W Linuksach, konieczne jest zainstalowanie pakietów libncurses i gcc z oficjalnych repozytoriów dystrybucji. W środowisku Apple MacOS X biblioteka i kompilator gcc instalowane są poprzez zainstalowanie darmowego środowiska Apple Xcode z App Store. Możliwe jest też zainstalowanie samych narzędzi konsolowych poprzez wywołanie w konsoli kompilatora gcc. W przypadku jego braku, wyświetlony zostanie komunikat o możliwości poprania i instalacji tego narzędzia.

2.2. MS Windows

Biblioteka curses w oryginalnej wersji nie działa w środowisku Microsoft Windows. Jest jednak kilka implementacji tej biblioteki na ten system operacyjny. Jedną z nich, używaną w niniejszej instrukcji, jest biblioteka PDCurses `http://pdcurses.sourceforge.net/`. Aby zainstalować tą bibliotekę w środowisku Code::Blocks należy pobrać skompilowaną wersję biblioteki dla środowiska MS Windows. Aktualna wersja 3.4 dostępna jest pod adresem `http://sourceforge.net/projects/pdcurses/files/pdcurses/3.4/pdc34dllw.zip/download`. W archiwum dostępne są cztery pliki, które należy skopiować do odpowiednich folderów kompilatora. Zakładając, że środowisko zainstalowane zostało w standardowej lokalizacji, należy przejść do folderu: `C:\Program Files (x86)\CodeBlocks\MinGW\`. Jeżeli lokalizacja jest inna, należy wybrać folder odpowiedni dla danej instalacji. Następnie plik biblioteki: `pdcurses.lib` należy skopiować do folderu `\lib`. Pliki nagłówkowe `curses.h` i `panel.h` należy skopiować do folderu `\include`. Ostatni plik `pdcurses.dll` należy skopiować do folderu `\bin`. To kończy instalację biblioteki.

3. Kompilacja

W niniejszej sekcji opisano sposób kompilacji programów wykorzystujących bibliotekę curses.

3.1. Kompilacja wCode::Blocks

W środowisku Code::Blocks należy przekazać kompilatorowi odpowiednią flagę. Z tego powodu konieczne jest pracowanie z wykorzystaniem projektów – nie da się kompilować pliku z jego pominięciem. Po utworzeniu projektu, należy zaznaczyć jego nazwę i otworzyć menu kontekstowe (najczęściej prawym przyciskiem myszy). Należy wybrać polecenie `Build options…`. W nowo otwartym oknie `Project build options` należy przejść do karty `Linker settings`. W polu `Other linker options` należy podać flagę odpowiednią do używanej biblioteki. W przypadku systemów uniksowych jest to `-lncurses`, a w przypadku MS Windows `-lpdcurses`. Na koniec potwierdzamy wybór przyciskiem `OK`.

3.2. Hello World!

Podstawowe elementy programu wykorzystującego bibliotekę `curses` przedstawione zostaną na przykładzie programu HelloWorld z Listingu 1.

```c
#include <curses.h>

int main(void)
{^^I
   initscr();    /* Przejście w tryb biblioteki curses */
   printw("Hello World !!!"); /* Wpisanie do bufora tekstu Hello World*/
   refresh();   /* Wyświetlenie bufora w oknie */
   getch();    /* Oczekiwanie na reakcję użytkownika */
   endwin();    /* Zakończenie trybu biblioteki curses*/

   return 0;
}
```

Listing 1: Program HelloWorld z wykorzystaniem biblioteki curses.

Funkcja `initscr()` uruchamia tryb pracy z biblioteką. Przed jej wywołaniem, program zachowuje się, jak zwykły program pracujący w konsoli. W trakcie uruchamiania trybu `curses` alokowana jest pamięć na obsługę głównego okna `stdscr`, które domyślnie wypełnia cały obszar konsoli.

Funkcja `printw()` podobnie jak funkcja `printf()` służy do wyświetlania tekstu. Działa jednak w trochę odmienny sposób. Funkcja w tej wersji wpisuje podany tekst do bufora powiązanego z oknem głównym. Tekst ten nie jest jeszcze widoczny dla użytkownika. Aby go wyświetlić konieczne jest odświeżenie okna poleceniem `refresh()`.

Funkcja `getch()` służy do pobierania znaku z klawiatury. Została tu jednak wykorzystana do zatrzymania programu przed jego zakończeniem.

Gdy biblioteka i wersja okienkowa nie jest już potrzebna lub program ma już być zakończony, należy wywołać funkcję `endwin()`. Powoduje ona zakończenie trybu okienkowego programu, przywrócenie terminala do normalnego trybu pracy, oraz powoduje zwolnienie pamięci wykorzystywanej na wewnętrzne dane biblioteki.

4. Funkcje inicjujące

Rozpoczynając tryb pracy z biblioteką, można też skonfigurować kilka dodatkowych parametrów pracy.

4.1. Funkcje

Standardowo, terminal buforuje wprowadzane znaki do czasu wstawienia znaku końca linii. Często jednak jest potrzebne wczytanie znaku od razu po jego wprowadzeniu (bez buforowania). Jest tak też w bibliotece `curses`, dlatego konieczne jest określenie, jak program ma reagować na wprowadzane znaki.

raw() Wszystkie znaki wprowadzane są bezpośrednio do programu. Polecenia zawieszenia działania (`CTRL-Z`) i przerwania i zakończenia (`CTRL-C`) są przekazywane do programu, ale nie generują sygnałów zawieszenia lub zakończenia dla systemu operacyjnego.

cbreak() Wszystkie znaki i wprowadzane również są bezpośrednio do programu. Jednakże polecenia zawieszenia działania (`CTRL-Z`) i przerwania i zakończenia (`CTRL-C`) generują odpowiednie sygnały zawieszenia lub zakończenia dla systemu operacyjnego.

echo() Wszystkie wprowadzane znaki wyświetlane są w standardowym oknie w czasie wpisywania.

noecho() Wszystkie wprowadzane znaki są wprowadzane do bufora i nie są widoczne od razu dla użytkownika. Funkcji tej używa się w celu zwiększenia kontroli, w którym miejscu ekranu ma zostać wyświetlona wprowadzona wartość. Wyświetlenie to musi być zaimplementowane osobno przez programistę.

keypad(nazwa_okna, TRUE) powoduje włączenie wczytywania przez program klawiszy funkcyjnych, takich jak `F1` itp.

halfdelay(time) Działa podobnie do `cbreak()`, ale znaki wprowadzone przez użytkownika są dla programu dostępne po upływie `time` dziesiątych sekundy.

4.2. Przykład

```c
#include <curses.h>

int main(void)
{
   int ch;

   initscr();   /* Inicjacja biblioteki */
   raw();   /* wyłączenie buforowania linii */
   keypad(stdscr, TRUE);   /* Włączenie wczytywania przycisków F1, F2 itd.*/
                                                /* dla głównego okna. */
   noecho();   /* Nie wyświetlaj wprowadzanych znaków. */

   printw("Napisz dowolny tekst, aby wypisać go pogubiona czcionka\n");
   ch = getch();   /* Jeżeli nie wywołano raw()
                        * konieczne będzie wciśnięcie Enter aby program
                        * kontynuował prace  */
   if(ch == KEY_F(1))^^I   /* Nie zadziała, jeżeli nie wywołano keypad()*/
      printw("F1 Key pressed");
   else
   {
      printw("The pressed key is ");
      attron(A_BOLD);
      printw("%c", ch);
      attroff(A_BOLD);
   }
   refresh();
   getch();
   endwin();

   return 0;
}
```

Listing 2: Program prezentujący konfigurowanie biblioteki curses.

5. Okna

Okien z biblioteki `curses` nie należy mylić z oknami znanymi z GUI współczesnych systemów operacyjnych. Przykładami mogą być takie aplikacje, jak Symantec Norton Commander, Midnight Commander, czy SUSE YaST (wersja tekstowa, nie YaST2 pracujący w środowisku graficznym). Okna możemy tu traktować, jako części ekranu, którymi możemy sterować w sposób niezależny. Daje to możliwość programiście dowolnego podzielenia ekranu na niezależne fragmenty. Wyświetlanie lub modyfikacja jednego z fragmentów pozostaje bez wpływu na działanie pozostałych części okna, co ułatwia projektowanie aplikacji.

Po uruchomieniu biblioteki, zawsze tworzone jest okno domyślne, dostępne przez zmienną globalną `stdscr`, które wypełnia całe okno konsoli.

5.0.1. Obszar roboczy

Położenie każdego elementu na ekranie określonej jest poprzez współrzędne kartezjańskie na płaszczyźnie (XY). Początek układu współrzędnych umieszczony jest w górnym lewym rogu okna. Przesuwając się wprawo, przesuwamy się wzdłuż osi OX, a przesuwając się w dół przesuwamy się wzdłuż osi OY. Obszar rysowania (które jest oknem głównym o rozmiarach ekranu lub konsoli) opisany jest poprzez strukturę SCREEN. Każde okno posiada swój opis w strukturze WINDOW. Opis obydwu struktur można znaleźć w pliku nagłówkowym `curses.h`. Nie należy ręcznie zmieniać wartości pól tych struktur.

5.0.2. Zmienne predefiniowane

LINES licza linii wyświetlanych na ekranie,

COLS liczba kolumn wyświetlanych na ekranie,

WINDOW *stdscr; Główne okno programu wypełniające cały ekran/całą konsolę.

5.0.3. Polecenia sterujące oknami

WINDOW *newwin(int nlines, int ncols, int begin_y, int begin_x); Zwraca wskaźnik na nowo utworzone okno lub NULL w przypadku błędu.`nlines` - liczba linii nowego okna, `ncols` - liczba kolumn nowego okna. Parametry `begin_y` i `begin_x` określają koordynaty lewego górnego rogu nowego okna.

int delwin(WINDOW *win); Usuwa wszystkie struktury powiązane z danym oknem. Nie powoduje wyczyszczenia ekranu.

int refresh(void); Powoduje odświeżenie zawartości całego ekranu

int wrefresh(WINDOW *win); Powoduje odświeżenie jedynie zawartości okna podanego jako parametr.

6. Funkcje wyświetlające dane

Do wyświetlania tekstu w bibliotece `curses` dostępne są trzy klasy funkcji:

1. `addch()`: Drukuje pojedynczy znak z atrybutami.
2. `printw()`: Drukuje sformatowany ciąg w sposób podobny do funkcji `printf()`.
3. `addstr()`: Drukuje ciąg.

Niektóre z wymienionych funkcji umożliwiają formatowanie tekstu poprzez nadanie tekstowi odpowiednich znaczników.

A_NORMAL Wyświetlanie tekstu bez ozdobników.

A_STANDOUT Zaznaczenie tekstu.

A_UNDERLINE Podkreślenie

A_REVERSE Odwrócenie kolorów

A_BLINK Mruganie

A_DIM Obraz przyciemniony

A_BOLD Pogrubienie

A_PROTECT Tryb ochronny

A_INVIS Widoczność napisów(widoczne czy ukryte)

A_ALTCHARSET Alternatywny zestaw znaków

A_CHARTEXT Mapa bitowa do uzyskiwania samego znaku bez parametrów

COLOR_PAIR(n) Para kolorów nr n.

Można też ustawić domyślne wartości parametrów poprzez użycie funkcji: `attron()` (dodaje kolejne atrybuty do już ustawionych) lub `attrset()` (kasuje stare ustawienia atrybutów i nadaje nowe). Jako parametry, obydwie funkcje przyjmują atrybut do ustawienia. W celu ustawienia kilku atrybutów, trzeba wypisać je wszystkie rozdzielone symbolem OR (|). Przykład:
`attron(A_REVERSE | A_BLINK);`

6.1. Funkcje klasy addch()

Funkcje z tej kategorii odpowiadają za wyświetlenie na ekranie jednego znaku.

int addch(const chtype ch); Wyświetla podany znak w głównym oknie w miejscu, gdzie aktualnie znajduje się kursor.

int waddch(WINDOW *win, const chtype ch); Wyświetla podany znak w podanym oknie w miejscu, gdzie aktualnie znajduje się kursor.

int mvaddch(int y, int x, const chtype ch); Wyświetla podany znak w oknie głównym na podanej pozycji.

int mvwaddch(WINDOW *win, int y, int x, const chtype ch); Wyświetla podany znak w podanym oknie na podanej pozycji.

6.2. Funkcje klasy printw()

Funkcje te służą do wyświetlania sformatowanych ciągów znaków w oknach biblioteki `curses`.

int printw(const char *fmt, ...); Wypisuje sformatowany ciąg w głównym oknie w miejscu, gdzie aktualnie znajduje się kursor.

int wprintw(WINDOW *win, const char *fmt, ...); Wypisuje sformatowany ciąg w podanym oknie w miejscu, gdzie aktualnie znajduje się kursor.

int mvprintw(int y, int x, const char *fmt, ...); Wypisuje sformatowany ciąg w głównym oknie w miejscu, które zostanie podane jako parametry funkcji.

int mvwprintw(WINDOW *win, int y, int x, const char *fmt, ...); Wypisuje sformatowany ciąg w podanym oknie w miejscu, które zostanie podane jako parametry funkcji.

6.3. Funkcje klasy addstr()

Poniższe funkcje służą do wyświetlania podanego ciągu.

int addstr(const char *str); Wyświetla wprowadzony napis w głównym oknie.

int addnstr(const char *str, int n); Wyświetla co najwyżej `n` znaków z wprowadzonego napisu w głównym oknie.

int waddstr(WINDOW *win, const char *str); Wyświetla wprowadzony napis w podanym oknie.

int waddnstr(WINDOW *win, const char *str, int n); Wyświetla co najwyżej `n` znaków z wprowadzonego napisu w podanym oknie.

int mvaddstr(int y, int x, const char *str); Wyświetla w oknie głównym podany napis w miejscu podanym jako parametry.

int mvaddnstr(int y, int x, const char *str, int n); Wyświetla co najwyżej `n` znaków z wprowadzonego napisu w oknie głównym w miejscu podanym jako parametry.

int mvwaddstr(WINDOW *win, int y, int x, const char *str); Wyświetla w podanym oknie podany napis w miejscu podanym jako parametry.

int mvwaddnstr(WINDOW *win, int y, int x, const char *str, int n); Wyświetla co najwyżej `n` znaków z wprowadzonego napisu w podanym oknie w miejscu podanym jako parametry.

7. Funkcje wczytujące dane

Do wczytywania tekstu w bibliotece `curses` dostępne są trzy klasy funkcji:

1. `getch()`: Wczytuje pojedynczy znak.
2. `scanw()`: Wczytuje sformatowany ciąg w sposób podobny do funkcji `scanf()`.
3. `getstr()`: Wczytuje ciąg.

7.1. Funkcje klasy getch()

int getch(void); odczytuje znak z klawiatury i umieszcza go w miejscu kursora w bieżącym oknie

int wgetch(WINDOW *win); odczytuje znak z klawiatury i umieszcza go w miejscu kursora w określonym oknie

int mvgetch(int y, int x); odczytuje znak z klawiatury i umieszcza go w pozycji `(y, x)`

int mvwgetch(WINDOW *win, int y, int x); odczytuje znak z klawiatury i umieszcza go w pozycji `(y, x)`

Odczytana wartość zwracana jest jako wynik funkcji. Jeżeli jest ona równa stałej `ERR` oznacza to, że operacja odczytu nie powiodła się. Klawisze funkcyjne można rozpoznać, porównując je do następujących stałych (Źródło: Manual BSD):

Nazwa stałej Nazwa przycisku
KEY_BREAK Break key
KEY_DOWN The four arrow keys ...
KEY_UP
KEY_LEFT
KEY_RIGHT
KEY_HOME Home key (upward+left arrow)
KEY_BACKSPACE Backspace
KEY_F0 Function keys; space for 64 keys is reserved.
KEY_F(n) For 0 <= n <= 63
KEY_DL Delete line
KEY_IL Insert line
KEY_DC Delete character
KEY_IC Insert char or enter insert mode
KEY_EIC Exit insert char mode
KEY_CLEAR Clear screen
KEY_EOS Clear to end of screen
KEY_EOL Clear to end of line
KEY_SF Scroll 1 line forward
KEY_SR Scroll 1 line backward (reverse)
KEY_NPAGE Next page
KEY_PPAGE Previous page
KEY_STAB Set tab
KEY_CTAB Clear tab

KEY_CATAB Clear all tabs
KEY_ENTER Enter or send
KEY_SRESET Soft (partial) reset
KEY_RESET Reset or hard reset
KEY_PRINT Print or copy
KEY_LL Home down or bottom (lower left)
KEY_A1 Upper left of keypad
KEY_A3 Upper right of keypad
KEY_B2 Center of keypad
KEY_C1 Lower left of keypad
KEY_C3 Lower right of keypad
KEY_BTAB Back tab key
KEY_BEG Beg(inning) key
KEY_CANCEL Cancel key
KEY_CLOSE Close key
KEY_COMMAND Cmd (command) key
KEY_COPY Copy key
KEY_CREATE Create key
KEY_END End key
KEY_EXIT Exit key
KEY_FIND Find key
KEY_HELP Help key
KEY_MARK Mark key
KEY_MESSAGE Message key
KEY_MOUSE Mouse event read
KEY_MOVE Move key
KEY_NEXT Next object key
KEY_OPEN Open key
KEY_OPTIONS Options key
KEY_PREVIOUS Previous object key
KEY_REDO Redo key
KEY_REFERENCE Ref(erence) key
KEY_REFRESH Refresh key
KEY_REPLACE Replace key
KEY_RESIZE Screen resized
KEY_RESTART Restart key
KEY_RESUME Resume key
KEY_SAVE Save key
KEY_SBEG Shifted beginning key
KEY_SCANCEL Shifted cancel key
KEY_SCOMMAND Shifted command key
KEY_SCOPY Shifted copy key
KEY_SCREATE Shifted create key
KEY_SDC Shifted delete char key
KEY_SDL Shifted delete line key
KEY_SELECT Select key
KEY_SEND Shifted end key
KEY_SEOL Shifted clear line key
KEY_SEXIT Shifted exit key
KEY_SFIND Shifted find key
KEY_SHELP Shifted help key
KEY_SHOME Shifted home key
KEY_SIC Shifted input key
KEY_SLEFT Shifted left arrowkey
KEY_SMESSAGE Shifted message key
KEY_SMOVE Shifted move key

KEY_SNEXT Shifted next key
KEY_SOPTIONS Shifted options key
KEY_SPREVIOUS Shifted prev key
KEY_SPRINT Shifted print key
KEY_SREDO Shifted redo key
KEY_SREPLACE Shifted replace key
KEY_SRIGHT Shifted right arrow
KEY_SRSUME Shifted resume key
KEY_SSAVE Shifted save key
KEY_SSUSPEND Shifted suspend key
KEY_SUNDO Shifted undo key
KEY_SUSPEND Suspend key
KEY_UNDO Undo key

Klawiatura numeryczna zorganizowana jest w następujący sposób:

A1	up	A3
left	B2	right
C1	down	C3

7.2. Funkcje klasy scanw()

int scanw(char *fmt, ...);

int wscanw(WINDOW *win, char *fmt, ...);

int mvscanw(int y, int x, char *fmt, ...);

int mvwscanw(WINDOW *win, int y, int x, char *fmt, ...);

int vw_scanw(WINDOW *win, char *fmt, va_list varglist);

int vwscanw(WINDOW *win, char *fmt, va_list varglist);

Funkcje `scanw()`, `wscanw()` i `mvscanw()` działają podobnie do funkcji `scanf()`. Pola, które nie są odwzorowane w `fmt` są tracone.

Funkcji `vw_scanw()` i `vwscanw()` można użyć, gdy ma zostać pobrana zmienna liczba argumentów.

7.3. Funkcje klasy getstr()

Funkcje tej klasy działają w sposób podobny do wielokrotnie wykonywanej funkcji z klasy `getch()`. Pobrany ciąg umieszczany jest w buforze przekazanym funkcji jako wskaźnik `str`.

int getstr(char *str);

int getnstr(char *str, int n);

int wgetstr(WINDOW *win, char *str);

int wgetnstr(WINDOW *win, char *str, int n);

int mvgetstr(int y, int x, char *str);

int mvwgetstr(WINDOW *win, int y, int x, char *str);

int mvgetnstr(int y, int x, char *str, int n);

int mvwgetnstr(WINDOW *, int y, int x, char *str, int n);

8. Kolory

Przed rozpoczęciem korzystania z trybu kolorów konieczne jest sprawdzenie, czy są one obsługiwane przez terminal. W tym celu należy skorzystać z poniższej funkcji `has_colors()`. Funkcja zwraca wartości `TRUE` lub `FALSE`.

Obsługa trybu kolorów rozpoczyna się od chwili wywołania funkcji `start_color()`. Biblioteka `curses` do zarządzania kolorami wykorzystuje tzw. pary kolorów, czyli kolor znaku i tła. Nie można używać tylko jednego z nich – obydwa muszą być zdefiniowane. Do definiowania pary kolorów służy funkcja `init_pair()`. Przyjmuje ona trzy parametry. Pierwszym jest numer identyfikujący zdefiniowaną parę kolorów w programie. Kolejnymi dwoma parametrami są kolory. Dostępne kolory zdefiniowane są jako stałe:

`COLOR_BLACK COLOR_RED COLOR_GREEN COLOR_YELLOW COLOR_BLUE COLOR_MAGENTA COLOR_CYAN COLOR_WHITE`

Przykład:

```
init_pair(1, COLOR_RED, COLOR_BLACK);
attron(COLOR_PAIR(1));
printw("Dowolny tekst");
attroff(COLOR_PAIR(1));
```

Listing 3: przykład użycia kolorów w bibliotece `curses`.

9. Zadania

Uwaga! Wszystkie programy muszą być napisane z podziałem na funkcje z parametrami!

1. Napisz program, który będzie symulował ekran logowania w systemie uniksowym, tzn. pozwoli wprowadzić login z echem, a hasło bez echa.

2. Napisz program, w którym podzielisz ekran na cztery równe części ułożone tak jak ćwiartki w dwuwymiarowym układzie współrzędnych. Po naciśnięciu przez użytkownika klawisza `Tab` (kod ASCII o wartości 9) program powinien narysować ramkę w pierwszym oknie, po kolejnym naciśnięciu w drugim oknie, itd. Po narysowaniu wszystkich ramek, jeśli użytkownik nadal będzie naciskał wymieniony klawisz to powinna zniknąć ramka z pierwszego okna, następnie drugiego itd. Program powinien działać ciągle, cyklicznie rysując i usuwając ramki z okien do momentu, aż użytkownik naciśnie klawisz `Esc` (kod ASCII o wartości 27).

3. Udoskonal program gry w życie rysując ramki wokół poszczególnych komórek. Zmniejsz rozmiar planszy, jeśli będzie ona kłopotliwa do wyświetlenia w oryginalnej wielkości.

4. Udoskonal program gry w życie, tak aby po naciśnięciu klawisza `F1` pojawiało się pod planszą okno, które będzie pozwalało wprowadzić nazwę pliku tekstowego, do którego program zapisze bieżący stan gry. Zmniejsz rozmiar planszy, jeśli będzie ona kłopotliwa do wyświetlenia w oryginalnej wielkości.

5. Udoskonal program gry w życie, tak, aby po naciśnięciu klawisza `F2` pod planszą pojawiało się okno, które pozwoli wprowadzić nazwę pliku tekstowego z zapisanym stanem gry. Po jej wprowadzeniu program powinien odczytać zwartość tego pliku i rozpocząć symulację od stanu w nim zawartego. Zmniejsz rozmiar planszy, jeśli będzie ona kłopotliwa do wyświetlenia w oryginalnej wielkości.

6. Udoskonal program gry w życie, tak aby po jego rozpoczęciu pojawiało się okno pozwalające użytkownikowi określić, czy symulacja ma się rozpocząć dla wzorca oscylatora (ang. *blinker*), krokodyla (ang. *ten in row*), czy też dla (pseudo)losowego rozmieszczenia żywych komórek.