Gra w życie w języku C++

Źródło: Michael Schiffer, domena publiczna.

Co wspólnego mają ze sobą pożary lasów, kształtowanie opinii publicznej oraz biofizyka? Ich wspólnym mianownikiem są automaty komórkowe, które wykorzystuje się do tworzenia modeli ilustrujących konkretne zjawiska.

Gra w życie jest jednym z najpopularniejszych automatów komórkowych. Implementacja opisującego ją algorytmu jest prosta. W tym e-materiale przygotujemy dwie wersje programu pozwalającego analizować ewolucję układu złożonego z komórek podlegających regułom *Gry w życie*.

Więcej informacji na temat tego algorytmu znajdziesz w e-materiale Gra w życie.

Z implementacją omawianego zagadnienia w pozostałych językach programowania możesz zapoznać się w e-materiałach:

- Gra w życie w języku Java,

- Gra w życie w języku Python.

> **Twoje cele**
>
> - Przeanalizujesz implementację *Gry w życie* w języku C++.
> - Prześledzisz, jak użyć biblioteki graficznej SDL do prezentacji przebiegu *Gry w życie*.
> - Zmodyfikujesz kod *Gry w życie* zgodnie z własnymi potrzebami.

Przeczytaj

Koncepcja programu

Zastanówmy się nad strukturą programu i poszczególnymi funkcjami, które chcielibyśmy zaimplementować. Rozpoczniemy od zainicjalizowania siatki komórek określonej wielkości oraz wypełnienia jej pierwszą, losowo wygenerowaną populacją.

Następnie utworzymy pętlę, dzięki której będziemy wypisywać zawartość siatki oraz przekształcać ją zgodnie z zasadami *Gry w życie*. Pseudokod głównej pętli naszego programu wygląda następująco:

```
siatka[n][n] ← {}
inicjalizuj_siatke()

Dopóki trwa gra wykonuj:
    wypisz_siatke()
    ewolucja()
```

Implementacja bez użycia biblioteki graficznej

Przejdźmy do implementacji automatu komórkowego. Zaczniemy od programu, w którym nie będziemy wykorzystywać biblioteki graficznej. Rezultaty działania wypiszemy w terminalu.

Zacznijmy od inicjalizacji kilku stałych globalnych.

```
const int ROZMIAR_SIATKI = 20;
const bool MARTWA = false;
const bool ZYWA = true;
```

Nasza siatka będzie mieć kształt kwadratu. Stała `ROZMIAR_SIATKI` przechowuje długość jego boku jako liczbę składających się nań komórek. Wartości `MARTWA` oraz

`ZYWA` są z kolei stałymi, które poprawią czytelność naszego kodu.

Siatką będzie dwuwymiarowa tablica zmiennych typu `bool`. Ponieważ każda komórka w *Grze w życie* może przyjąć tylko jeden z dwóch stanów, ten typ danych idealnie nadaje się do reprezentacji planszy.

```
bool siatka[ROZMIAR_SIATKI][ROZMIAR_SIATKI];
```

Dodatkowo poza funkcją główną `main()` zdefiniujemy następujące funkcje:

- `inicjalizuj_siatke()` – funkcja odpowiedzialna za losowe wygenerowanie pierwszej populacji komórek,
- `policz_zywych_sasiadow()` – funkcja odpowiedzialna za policzenie żywych sąsiadów komórki,
- `ewolucja()` – funkcja realizująca jeden cykl ewolucji komórek,
- `wypisz_siatke()` – funkcja wypisująca obecny stan planszy na standardowe wyjście.

Do wypisywania obecnego stanu planszy wykorzystamy znaki z zestawu Unicode. Sprawdzą się one o wiele lepiej od standardowego zestawu znaków ASCII podczas wyświetlania zawartości komórek w terminalu.

Aby wyświetlać znaki Unicode, musimy odpowiednio ustawić tryb wypisywania.

```
_setmode(_fileno(stdout), _O_U16TEXT);
```

Pełny kod *Gry w życie* w języku C++ wygląda następująco:

```
// Dołączenie potrzebnych nam bibliotek
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <Windows.h>
```

```
#include <io.h>
#include <fcntl.h>

// Inicjalizacja globalnych stałych
const int ROZMIAR_SIATKI = 20;
const bool MARTWA = false;
const bool ZYWA = true;

// Prototypy funkcji
void inicjalizuj_siatke(bool siatka[][ROZMIAR_SIATKI]);
void wypisz_siatke(bool siatka[][ROZMIAR_SIATKI]);
void ewolucja(bool siatka[][ROZMIAR_SIATKI]);
int policz_zywych_sasiadow(int y, int x, bool siatka[][ROZMIAR_S


int main() {
    _setmode(_fileno(stdout), _O_U16TEXT);
    bool siatka[ROZMIAR_SIATKI][ROZMIAR_SIATKI];
    inicjalizuj_siatke(siatka);

    // Główna pętla naszej gry
    while (true) {
        wypisz_siatke(siatka);
        ewolucja(siatka);

        /* Jako ze program wykonuje sie bardzo szybko to
        żebyśmy mogli zaobserwować co się dzieje musimy
        go wstrzymać na pół sekundy przy każdej
        iteracji pętli */
        Sleep(500);
    }
}

void inicjalizuj_siatke(bool siatka[][ROZMIAR_SIATKI]) {
    /* Do wygenerowania pierwszej populacji użyjemy
    generatora liczb pseudolosowych rand() */
```

```
    srand(time(NULL));

    int temp;

    for (int i = 0; i < ROZMIAR_SIATKI; i++) {
        for (int j = 0; j < ROZMIAR_SIATKI; j++) {
            temp = rand();

            if (temp % 5 == 0)
                siatka[i][j] = ZYWA;
            else
                siatka[i][j] = MARTWA;
        }
    }
}

void ewolucja(bool siatka[][ROZMIAR_SIATKI]) {
    /* Aby poprawnie wykonać ewolucje najpierw dla każdej
    komórki obliczamy liczbe jej żywych sąsiadów,
    a następnie zgodnie z regułami zmieniamy
    stan każdej komórki */
    int zywi_sasiedzi[ROZMIAR_SIATKI][ROZMIAR_SIATKI];

    for (int i = 0; i < ROZMIAR_SIATKI; i++) {
        for (int j = 0; j < ROZMIAR_SIATKI; j++) {
            zywi_sasiedzi[i][j] = policz_zywych_sasiadow(i, j, s
        }
    }

    for (int i = 0; i < ROZMIAR_SIATKI; i++) {
        for (int j = 0; j < ROZMIAR_SIATKI; j++) {
            if (siatka[i][j] == MARTWA && zywi_sasiedzi[i][j] ==
                siatka[i][j] = ZYWA;
            else if (siatka[i][j] == ZYWA)
                if (zywi_sasiedzi[i][j] < 2 || zywi_sasiedzi[i][
                    siatka[i][j] = MARTWA;
```

```
        }
    }
}

int policz_zywych_sasiadow(int y, int x, bool siatka[][ROZMIAR_S
    int ile = 0;

    for (int i = y - 1; i <= y + 1; i++) {
        for (int j = x - 1; j <= x + 1; j++) {
            if (i >= 0 && j >= 0 && i < ROZMIAR_SIATKI && j < RO
                ile++;
        }
    }

    /* Jako ze w powyższej pętli mogliśmy dodać wartość
    komórki dla której sprawdzamy to musimy to odjać */
    if (siatka[y][x] == ZYWA)
        ile--;

    return ile;
}

void wypisz_siatke(bool siatka[][ROZMIAR_SIATKI]) {
    // Na samym początku czyścimy konsole
    system("cls");

    /* Teraz wypisujemy obecny stan siatki
    wykorzystujac znak czarnego kwadratu o
    numerze Unicode 0x25A0 */
    for (int i = 0; i < ROZMIAR_SIATKI; i++) {
        for (int j = 0; j < ROZMIAR_SIATKI; j++) {
            if (siatka[i][j] == ZYWA)
                wprintf(L"\x25a0");
            else
                wprintf(L" ");
        }
```

```
114
115         wprintf(L"\n");
116     }
117 }
```

Słownik

> ***Simple DirectMedia Layer***
>
> biblioteka służąca do tworzenia grafik i programów multimedialnych; zapewnia dostęp do sprzętu audio, klawiatury, myszy oraz sprzętu graficznego; jest obsługiwana przez większość systemów operacyjnych

> **zestaw znaków *Unicode***
>
> zestaw znaków definiujący wszystkie znaki pisarskie, używane na świecie

Symulacja interaktywna

Polecenie 1

Zapoznaj się z symulacją interaktywną *Gry w życie*. W narzędziu można zmieniać kolor komórek w każdym kroku, a także obserwować przebieg gry. Przetestuj narzędzie dla różnych aparatów komórkowych.

Implementacja z użyciem biblioteki graficznej

Polecenie 2

Zapoznaj się z prezentacją multimedialną przedstawiającą implementację *Gry w życie* z użyciem biblioteki graficznej.

1

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1F79CHYI

Prezentacja siatki komórek przy użyciu znaków *Unicode* w terminalu nie jest zbyt estetyczną formą przedstawienia działania automatu komórkowego. Napiszemy zatem inną wersję programu, tym razem używając zewnętrznej biblioteki graficznej

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1F79CHYI

Biblioteka SDL jest bardzo popularną biblioteką graficzną, wykorzystywaną podczas pisania programów w języku C++. Jest ona łatwa

w użyciu, co czyni ją idealnym narzędziem do graficznego przedstawiania procesu ewolucji układu komórek. Dostępne są dwie wersje biblioteki: SDL 1.2 oraz SDL 2.0. My wykorzystamy wersję 2.0.

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1F79CHYI

Większa część przedstawionego wcześniej kodu się nie zmieni. Zmodyfikujemy jedynie funkcję `wypisz\_siatke()` oraz wprowadzimy kilka nowych stałych globalnych.

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1F79CHYI

Dodajemy biblioteki.

```
1
2 // Dołączenie potrzebnych
  nam bibliotek
3 #include <SDL.h>
4 #include <iostream>
5 #include <stdlib.h>
6 #include <time.h>
7 #include <Windows.h>
8
```

5

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1F79CHYI

Inicjalizujemy stałe globalne.

```

// Inicjalizacja stałych globalnych
const int ROZMIAR_EKRANU = 800; // rozmiar ekranu w pikselach
const int ROZMIAR_SIATKI = 50;
const int ROZMIAR_KOMORKI = ROZMIAR_EKRANU / ROZMIAR_SIATKI;
const bool MARTWA = false;
const bool ZYWA = true;

SDL_Renderer* renderer = NULL; // obiekt rysujacy
SDL_Window* okno = NULL; // okno gry

```

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1F79CHYI

Tworzymy prototypy funkcji.

```

void inicjalizuj_siatke(bool siatka[][ROZMIAR_SIATKI]);
void wypisz_siatke(bool siatka[][ROZMIAR_SIATKI]);
void ewolucja(bool siatka[][ROZMIAR_SIATKI]);
int policz_zywych_sasiadow(int y, int x, bool siatka[][ROZMIAR_SIATKI]);

```

7

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1F79CHYI

Tworzymy funkcję `main`.

```
1
2 int main(int argc, char*
  argv[]) {
3 }
4
```

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1F79CHYI

W środku funkcji `main` tworzymy okno aplikacji.

```
1
2 int main(int argc, char*
  argv[]) {
3     /* Na samym poczatku
  tworzymy okno aplikacji
4     w którym będziemy
  rysować nasza siatke */
5
  SDL_Init(SDL_INIT_VIDEO);
6
  SDL_CreateWindowAndRendere
  r(ROZMIAR_EKRANU,
  ROZMIAR_EKRANU, 0, &okno,
  &renderer);
7
8     bool
  siatka[ROZMIAR_SIATKI]
  [ROZMIAR_SIATKI];
```

```
9
    inicjalizuj_siatke(siatka)
    ;
10
11      while (true) {
12
    wypisz_siatke(siatka);
13          ewolucja(siatka);
14
15          /* Tak jak
    poprzednim razem,
    wstrzymujemy program
16          na pół sekundy, po
    każdej iteracji pętli */
17          Sleep(500);
18      }
19
20      return 0;
21  }
22
```

9

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1F79CHYI

Generujemy pierwszą populację.

```
1
2   void
    inicjalizuj_siatke(bool
    siatka[][ROZMIAR_SIATKI])
    {
3       /* Do wygenerowania
    pierwszej populacji
    użyjemy
4       generatora liczb
    pseudolosowych rand() */
5       srand(time(NULL));
6
7       int temp;
8
```

```
    for (int i = 0; i < ROZMIAR_SIATKI; i++) {
        for (int j = 0; j < ROZMIAR_SIATKI; j++) {
            temp = rand();

            if (temp % 5 == 0)
                siatka[i][j] = ZYWA;
            else
                siatka[i][j] = MARTWA;
        }
    }
}

```

10

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1F79CHYI

Implementujemy ewolucję.

```

void ewolucja(bool siatka[][ROZMIAR_SIATKI])
{
    /* Aby poprawnie wykonać ewolucje najpierw dla każdej
    komórki obliczamy liczbe jej żywych sąsiadów,
    a następnie, zgodnie z regulami zmieniamy
    stan każdej komórki */
    int zywi_sasiedzi[ROZMIAR_SIATKI][ROZMIAR_SIATKI];

```

```
    for (int i = 0; i < ROZMIAR_SIATKI; i++) {
        for (int j = 0; j < ROZMIAR_SIATKI; j++) {
            zywi_sasiedzi[i][j] = policz_zywych_sasiadow(i, j, siatka);
        }
    }

    for (int i = 0; i < ROZMIAR_SIATKI; i++) {
        for (int j = 0; j < ROZMIAR_SIATKI; j++) {
            if (siatka[i][j] == MARTWA && zywi_sasiedzi[i][j] == 3)
                siatka[i][j] = ZYWA;
            else if (siatka[i][j] == ZYWA)
                if (zywi_sasiedzi[i][j] < 2 || zywi_sasiedzi[i][j] > 3)
                    siatka[i][j] = MARTWA;
        }
    }
}

int policz_zywych_sasiadow(int y, int x, bool siatka[][ROZMIAR_SIATKI]) {
    int ile = 0;

    for (int i = y - 1; i <= y + 1; i++) {
        for (int j = x - 1; j <= x + 1; j++) {
            if (i >= 0 && j >= 0 && i < ROZMIAR_SIATKI && j <
```

```
            ROZMIAR_SIATKI &&
            siatka[i][j] == ZYWA)
32                  ile++;
33          }
34      }
35
36      /* Jako że w
        przedstawionej pętli
        mogliśmy dodać wartość
37      komórki dla której
        sprawdzamy, to musimy to
        odjać */
38      if (siatka[y][x] ==
        ZYWA)
39          ile--;
40
41      return ile;
42  }
43
```


Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1F79CHYI

Ustawiamy odpowiednie kolory i rysujemy linie siatki.

```
1
2   void wypisz_siatke(bool
    siatka[ROZMIAR_SIATKI]
    [ROZMIAR_SIATKI]) {
3       // Ustawiamy kolor
    rysownika na biały
4
    SDL_SetRenderDrawColor(ren
    derer, 255, 255, 255,
    SDL_ALPHA_OPAQUE);
5
6       // Czyścimy okno
    aplikacji, zamalowujac je
    na biało
```

```
7
SDL_RenderClear(renderer);
8
9       // Ustawiamy kolor
rysownika na szary
10
SDL_SetRenderDrawColor(ren
derer, 190, 190, 190,
SDL_ALPHA_OPAQUE);
11
12      // Rysujemy linie
naszej siatki
13      for (int i = 0; i <
ROZMIAR_SIATKI; i++) {
14
SDL_RenderDrawLine(rendere
r,
15                  0,
ROZMIAR_KOMORKI * i,
16
ROZMIAR_EKRANU,
ROZMIAR_KOMORKI * i);
17
18
SDL_RenderDrawLine(rendere
r,
19
ROZMIAR_KOMORKI * i, 0,
20
ROZMIAR_KOMORKI * i,
ROZMIAR_EKRANU);
21      }
22
23      // Ustawiamy kolor
rysownika na czarny
24
SDL_SetRenderDrawColor(ren
derer, 0, 0, 0,
SDL_ALPHA_OPAQUE);
25
26      /* Zamalowujemy na
czarno kwadraty w naszej
siatce,
27              reprezentujące
żywe komórki  */
```

```
28      for (int i = 0; i < ROZMIAR_SIATKI; i++) {
29          for (int j = 0; j < ROZMIAR_SIATKI; j++) {
30              if (siatka[i][j] == ZYWA) {
31                  SDL_Rect r = { j * ROZMIAR_KOMORKI, i * ROZMIAR_KOMORKI, ROZMIAR_KOMORKI, ROZMIAR_KOMORKI };
32                  SDL_RenderFillRect(renderer, &r);
33              }
34          }
35      }
36
37      // Aktualizujemy zawartość naszego okna
38      SDL_RenderPresent(renderer);
39  }
40
```

12

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/P1F79CHYI

Cały kod programu:

```
1
2   // Dołączenie potrzebnych nam bibliotek
3   #include <SDL.h>
4   #include <iostream>
5   #include <stdlib.h>
6   #include <time.h>
7   #include <Windows.h>
8
```

```
// Inicjalizacja stałych globalnych
const int ROZMIAR_EKRANU = 800; // rozmiar ekranu w pikselach
const int ROZMIAR_SIATKI = 50;
const int ROZMIAR_KOMORKI = ROZMIAR_EKRANU / ROZMIAR_SIATKI;
const bool MARTWA = false;
const bool ZYWA = true;

SDL_Renderer* renderer = NULL; // obiekt rysujacy
SDL_Window* okno = NULL; // okno gry

// Prototypy funkcji
void inicjalizuj_siatke(bool siatka[][ROZMIAR_SIATKI]);
void wypisz_siatke(bool siatka[][ROZMIAR_SIATKI]);
void ewolucja(bool siatka[][ROZMIAR_SIATKI]);
int policz_zywych_sasiadow(int y, int x, bool siatka[][ROZMIAR_SIATKI]);

int main(int argc, char* argv[]) {
    /* Na samym poczatku tworzymy okno aplikacji,
    w którym będziemy rysować nasza siatke */

SDL_Init(SDL_INIT_VIDEO);

SDL_CreateWindowAndRender
```

```
er(ROZMIAR_EKRANU,
ROZMIAR_EKRANU, 0, &okno,
&renderer);

    bool
siatka[ROZMIAR_SIATKI]
[ROZMIAR_SIATKI];

inicjalizuj_siatke(siatka
);

    while (true) {

wypisz_siatke(siatka);
        ewolucja(siatka);

        /* Tak jak
poprzednim razem,
wstrzymujemy program
        na pół sekundy,
po każdej iteracji pętli
*/
        Sleep(500);
    }

    return 0;
}

void
inicjalizuj_siatke(bool
siatka[][ROZMIAR_SIATKI])
{
    /* Do wygenerowania
pierwszej populacji
użyjemy
    generatora liczb
pseudolosowych rand() */
    srand(time(NULL));

    int temp;

    for (int i = 0; i <
ROZMIAR_SIATKI; i++) {
        for (int j = 0; j
< ROZMIAR_SIATKI; j++) {
```

```
            temp =
rand();

            if (temp % 5
== 0)
                siatka[i]
[j] = ZYWA;
            else
                siatka[i]
[j] = MARTWA;
        }
    }
}

void ewolucja(bool
siatka[][ROZMIAR_SIATKI])
{
    /* Aby poprawnie
wykonać ewolucje najpierw
dla każdej
    komórki obliczamy
liczbe jej żywych
sąsiadów,
    a następnie, zgodnie
z regulami, zmieniamy
    stan każdej komórki
*/
    int
zywi_sasiedzi[ROZMIAR_SIA
TKI][ROZMIAR_SIATKI];

    for (int i = 0; i <
ROZMIAR_SIATKI; i++) {
        for (int j = 0; j
< ROZMIAR_SIATKI; j++) {

zywi_sasiedzi[i][j] =
policz_zywych_sasiadow(i,
j, siatka);
        }
    }

    for (int i = 0; i <
ROZMIAR_SIATKI; i++) {
```

```
79          for (int j = 0; j
< ROZMIAR_SIATKI; j++) {
80              if (siatka[i]
[j] == MARTWA &&
zywi_sasiedzi[i][j] == 3)
81                  siatka[i]
[j] = ZYWA;
82              else if
(siatka[i][j] == ZYWA)
83                  if
(zywi_sasiedzi[i][j] < 2
|| zywi_sasiedzi[i][j] >
3)
84
siatka[i][j] = MARTWA;
85          }
86      }
87  }
88
89  int
policz_zywych_sasiadow(in
t y, int x, bool siatka[]
[ROZMIAR_SIATKI]) {
90      int ile = 0;
91
92      for (int i = y - 1; i
<= y + 1; i++) {
93          for (int j = x -
1; j <= x + 1; j++) {
94              if (i >= 0 &&
j >= 0 && i <
ROZMIAR_SIATKI && j <
ROZMIAR_SIATKI &&
siatka[i][j] == ZYWA)
95                  ile++;
96          }
97      }
98
99      /* Jako że w
powyższej pętli mogliśmy
dodać wartość
100     komórki dla której
sprawdzamy, to musimy to
odjać */
```

```
    if (siatka[y][x] == ZYWA)
        ile--;

    return ile;
}

void wypisz_siatke(bool siatka[ROZMIAR_SIATKI][ROZMIAR_SIATKI]) {
    // Ustawiamy kolor rysownika na biały
    SDL_SetRenderDrawColor(renderer, 255, 255, 255, SDL_ALPHA_OPAQUE);

    // Czyścimy okno aplikacji, zamalowujac je na biało
    SDL_RenderClear(renderer);

    // Ustawiamy kolor rysownika na szary
    SDL_SetRenderDrawColor(renderer, 190, 190, 190, SDL_ALPHA_OPAQUE);

    // Rysujemy linie naszej siatki
    for (int i = 0; i < ROZMIAR_SIATKI; i++) {
        SDL_RenderDrawLine(renderer,
                           0, ROZMIAR_KOMORKI * i,
                           ROZMIAR_EKRANU, ROZMIAR_KOMORKI * i);

```

```
123 
    SDL_RenderDrawLine(render
    er,
124 
    ROZMIAR_KOMORKI * i, 0,
125 
    ROZMIAR_KOMORKI * i,
    ROZMIAR_EKRANU);
126     }
127 
128     // Ustawiamy kolor
    rysownika na czarny
129 
    SDL_SetRenderDrawColor(re
    nderer, 0, 0, 0,
    SDL_ALPHA_OPAQUE);
130 
131     /* Zamalowujemy na
    czarno kwadraty w naszej
    siatce,
132     reprezentujace żywe
    komórki */
133     for (int i = 0; i <
    ROZMIAR_SIATKI; i++) {
134         for (int j = 0; j
    < ROZMIAR_SIATKI; j++) {
135             if (siatka[i]
    [j] == ZYWA) {
136                 SDL_Rect
    r = { j *
    ROZMIAR_KOMORKI, i *
    ROZMIAR_KOMORKI,
    ROZMIAR_KOMORKI,
    ROZMIAR_KOMORKI };
137 
    SDL_RenderFillRect(render
    er, &r);
138             }
139         }
140     }
141 
142     // Aktualizujemy
    zawartość naszego okna
143 
    SDL_RenderPresent(rendere
```

```
r);
}

```

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Zaznacz, jakiej biblioteki graficznej użyliśmy do wizualizacji *Gry w życie*.

- ◯ OpenGL
- ◯ SIGIL
- ◯ SDL
- ◯ SFML

Ćwiczenie 2

Uporządkuj kolejne kroki pseudokodu głównej pętli *Gry w życie*.

```
inicjalizuj_siatke()
    wypisz_siatke()
Dopóki trwa gra wykonuj:
    ewolucja()
siatka[n][n] ← {}
```

Ćwiczenie 3

Wskaż, czym jest zestaw znaków *Unicode*.

- ◯ Zestawem znaków zawierającym jedynie litery alfabetu łacińskiego, cyfry, znaki przestankowe i polecenia sterujące.
- ◯ Zestawem znaków obejmującym wszystkie pisma używane na świecie.
- ◯ Zestawem znaków obejmującym wszystkie pisma używane na świecie poza znakami chińskimi.

Ćwiczenie 4

Połącz w pary nazwy funkcji z ich opisem.

`rand()`	funkcja wstrzymująca działanie programu na określoną liczbę milisekund
`system("cls")`	funkcja wykonująca systemowe polecenie czyszczenia konsoli
`wprintf()`	funkcja generująca liczby pseudolosowe
`Sleep()`	funkcja mogąca wypisać, na standardowe wyjście, znaki ze zbioru *Unicode*

Ćwiczenie 5

Wskaż linię kodu odpowiedzialną za ustawienie trybu wypisywania programu tak, aby mógł on wypisać na standardowe wyjście znaki *Unicode*.

- ◯ `_set(_fileno(stdout), _O_U16TEXT);`
- ◯ `_setmode(_fileno(stdout), _O_U4TEXT);`
- ◯ `_setmode(_fileno(stdin), _O_U16TEXT);`
- ◯ `_setmode(_fileno(stdout), _O_U16TEXT);`

Ćwiczenie 6

Uzupełnij ciało funkcji `ewolucja()`, która zajmuje się przeprowadzaniem pojedynczego cyklu ewolucji komórek.

```
int zywi_sasiedzi[ROZMIAR_SIATKI][ROZMIAR_SIATKI];

for (int i = 0; i < ROZMIAR_SIATKI; i++) {
    for (int j = 0; j < ROZMIAR_SIATKI; j++) {
        zywi_sasiedzi[i][j] = [          ]
    }
}

for (int i = 0; i < ROZMIAR_SIATKI; i++) {
    for (int j = 0; j < ROZMIAR_SIATKI; j++) {
        if (siatka[i][j] == && zywi_sasiedzi[i][j] == [          ]
)
            siatka[i][j] = [          ];

        else if (siatka[i][j] == ZYWA)
            if (zywi_sasiedzi[i][j] [          ] 2 ||
zywi_sasiedzi[i][j] [          ] 3)
                siatka[i][j] = [          ];
    }
}
```

`>` `policz_zywych_sasiadow(i, j, siatka);` `<` `MARTWA` `ZYWA` `3`

Ćwiczenie 7

Wskaż funkcję odpowiadającą za zmianę koloru rysownika w bibliotece graficznej SDL.

- ◯ `SDL_SetRenderColor()`
- ◯ `SDL_SetRendererDrawColor()`
- ◯ `SDL_SetRenderDrawColor()`
- ◯ `SFML_SetRenderDrawColor()`

Ćwiczenie 8

Uzupełnij ciało funkcji `polcz_zywych_sasiadow()`, która zajmuje się zwróceniem liczby żywych sąsiadów danej komórki.

```
int policz_zywych_sasiadow(int y, int x, bool siatka[]
[ROZMIAR_SIATKI]) {
    [          ] ile = 0;

    for (int i = [          ]; i <= y + 1; i++) {
        for (int j = x - 1; j <= [          ]; j++) {
            if (i >= 0 && j >= 0 && i < ROZMIAR_SIATKI && j <
ROZMIAR_SIATKI && siatka[i][j] == [          ])
                    [          ];
        }
    }

    if (siatka[y][x] == ZYWA)
        [          ];

    return [          ];
}
```

`bool` `x + 1` `ile++` `ile--` `ZYWA` `x-1` `y - 1` `int` `MARTWA` `siatka` `string` `y+1` `ile`

Dla nauczyciela

Autor: Bartosz Zadrożny

Przedmiot: Informatyka

Temat: Gra w życie w języku C++

Grupa docelowa:

Liceum ogólnokształcące i technikum, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

> II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.
>
> III. Posługiwanie się komputerem, urządzeniami cyfrowymi i sieciami komputerowymi, w tym: znajomość zasad działania urządzeń cyfrowych i sieci komputerowych oraz wykonywania obliczeń i programów.
>
> IV. Rozwijanie kompetencji społecznych, takich jak: komunikacja i współpraca w grupie, w tym w środowiskach wirtualnych, udział w projektach zespołowych oraz zarządzanie projektami.

Treści nauczania – wymagania szczegółowe

> II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) projektuje i tworzy rozbudowane programy w procesie rozwiązywania problemów, wykorzystuje w programach dobrane do algorytmów struktury danych, w tym struktury dynamiczne i korzysta z dostępnych bibliotek dla tych struktur;

2) stosuje zasady programowania strukturalnego i obiektowego w rozwiązywaniu problemów;

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

IV. Rozwijanie kompetencji społecznych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

2) analizuje i charakteryzuje wpływ trendów w historycznym rozwoju pojęć, metod informatyki oraz technologii na możliwości rozwiązywania problemów teoretycznych i praktycznych;

3) przygotowuje się do świadomego wyboru kierunku i zakresu dalszego kształcenia, głównie informatycznego, z myślą o przyszłej karierze zawodowej.

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Przeanalizujesz implementację *Gry w życie* w języku C++.

- Prześledzisz, jak użyć biblioteki graficznej SDL do prezentacji przebiegu *Gry w życie*.
- Zmodyfikujesz kod *Gry w życie* zgodnie z własnymi potrzebami.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka C++, w tym kompilator GCC/G++ 4.5 (lub nowszej wersji) i Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Przebieg lekcji

Przed lekcją:

1. Nauczyciel prosi chętne lub wybrane osoby o przygotowanie krótkich wystąpień dotyczących struktur wykorzystywanych w grze w życie.
2. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Gra w życie w języku C++". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Wyświetlenie przez nauczyciela tematu i celów zajęć, przejście do wspólnego ustalenia kryteriów sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. Uczniowie analizują przykład z sekcji „Przeczytaj" i powtarzają zaprezentowane rozwiązanie na swoim komputerze.
2. **Praca z multimedium.** Uczniowie pracują w parach. Analizują treści zawarte w sekcji „Symulacja multimedialna".
3. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenia nr 1-8 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

Faza podsumowująca:

1. Nauczyciel zadaje pytania podsumowujące, np.
2. Na koniec zajęć z programowania w C++ nauczyciel prosi uczniów o rozwinięcie zdania: „Na dzisiejszych zajęciach nauczyłam/łem się jak…".

Praca domowa:

1. Uczniowie testują za pomocą symulacji działanie różnych aparatów komórkowych i zapisują wnioski.

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka C++.
- Oficjalna dokumentacja techniczna dla kompilatora GCC/G++ 4.5 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać treści w sekcjach: „Przeczytaj”, „Symulacja interaktywna”, „Sprawdź się” jako materiał do lekcji powtórkowej.