Tomasz Jaśniewski

C++ ZADANIA ZAAWANSOWANE

Helion

Redaktor prowadzący: Maciej Olanicki, Małgorzata Kulik
Projekt okładki: Studio Gravite / Olsztyn Obarek,
Pokoński, Pazdrijowski, Zaprucki

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/cppzaz
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Materiały do książki znaleźć można pod adresem:
https://ftp.helion.pl/przyklady/cppzaz.zip

ISBN: 978-83-289-0181-0

Printed in Poland.

Spis treści

KATEGORIA PIĄTA #5

Nerd (25 zadań)

Umiejętności z tej kategorii zawierają wszystkie wcześniej sugerowane umiejętności.

Sugerowane, przydatne umiejętności:

- znajomość biblioteki graficznej 2D, np. SFML 2.5.1 lub nowszej/innej; przykładowe zadania będą rozwiązywane za pomocą biblioteki SFML dla VS2022 pobranej przez menedżer pakietów NuGet;
- dziedziczenie klas, sekcje `public`, `protected` i `private` w definicji klas;
- wykorzystywanie wielu plików *.cpp*, *.hpp* w tworzeniu aplikacji;
- podstawowe użycie szablonów `template`;
- wielowątkowość (proste użycie);
- proste programowanie sieciowe (połączenie protokołem TCP/UDP i przy użyciu gniazd);
- tworzenie makro za pomocą dyrektyw `#define`;
- funkcje z rodziny `printf` (`printf`, `sprintf`, `sprintf_s` itp.);
- odczytywanie aktualnej godziny/minuty/sekundy (przydatne biblioteki `<chrono>`, `<format>`);
- przydatne zagadnienia: iteratory wstawiające i strumienia, składowe kontenerów, definicje typów, cechy typowe, predykaty, folding.

Uwaga! Wykorzystując wyżej wymienione umiejętności, możesz ponownie rozwiązać niektóre zadania z wcześniejszych kategorii. Wiele z poniższych zadań można rozwiązać bez stosowania sugerowanych umiejętności.

5.1 [4:2,2] (2D)

Utwórz klasę `Kolo`, która będzie reprezentować koła o promieniu *r*. Uznajemy, że między obiektami klasy zachodzi relacja porządkująca, polegająca na tym, że koło o mniejszym promieniu jest mniejsze. Zaprojektuj mechanizm sortowania kół metodą **sortowania bąbelkowego**[17] **[2,]** i zwizualizuj go w dowolnym trybie graficznym 2D. **[2]**

[17] Jedno z najprostszych sortowań o marnej złożoności $O(n^2)$. Niemniej jest bardzo proste, a sercem tego algorytmu jest porównywanie sąsiednich par elementów i zamienianie ich miejscami zgodnie z oczekiwanym porządkiem.

Aby ułatwić wizualizację, przyjmij, że maksymalna liczba losowo utworzonych kół do posortowania wynosi 20, ale nie mniej niż 10. Ich losowy promień nie powinien być większy niż 100 i nie mniejszy niż 10, a wartości promieni powinny dzielić się przez 10.

5.2 [8:3,3,2] (2D)

W pliku *2_labirynt.txt* znajduje się zapisany tekstowo labirynt, reprezentowany przez odpowiednie znaki. Znak # jest nieprzekraczalną ścianą, a znak . (kropka) polem, po którym można się poruszać. Nie można poruszać się po skosie, lecz tylko w kierunkach: góra, dół, lewo, prawo. Pole oznaczone znakiem *S* to pozycja startowa. Pole oznaczone znakiem *K* to miejsce, do którego trzeba dojść. Zastosuj dowolny algorytm wyszukiwania drogi[18] **[3,]**, a następnie zaprezentuj jego działanie w trybie graficznym. Zwróć szczególną uwagę na ukazanie ścieżki opuszczenia labiryntu. Postaraj się, by była to droga jak najkrótsza! **[3,]** Oprócz ukazania drogi świetnie byłoby zwizualizować, jak algorytm szuka ścieżki, analizuje przestrzeń labiryntu, zanim odszuka drogę łączącą pole startowe i końcowe. **[2]**

5.3 [3] (2D)

Stwórz program rysujący na płaszczyźnie wykres funkcji wielomianowej do maksymalnie trzeciego stopnia. Program powinien pozwalać wprowadzać wzór takiej funkcji, a następnie prezentować jej wykres. Możemy przyjąć, że jeden piksel obrazu odpowiada wartości 1 na osi XY. Do obliczeń wartości w punkcie wykorzystaj schemat Hornera[19]. Zaprezentuj wykresy trzech funkcji:

$-0{,}05x^3 + 0{,}5x^2 - 2x - 140$, $0{,}1x^2 - 4x - 44$ oraz $-x + 120$ w przedziale $-500:500$. **[3]**

5.4 [3:1,2] (2D)

W pliku *4_pozycje.txt* znajdują się pozycje kół na płaszczyźnie, długości promieni oraz kierunek ich ruchu (współrzędne środka koła, promień koła i stopnie od 0 do 359, przy założeniu, że 0 stopni to kierunek „północny"). Wszystkie koła ruszają się z prędkością 10 punktów na sekundę (przez punkt rozumiem jednostkę opisującą odległości na płaszczyźnie na scenie środowiska 2D. Przykładowo, gdy chcę pewien obiekt leżący w punkcie [0, 0] przesunąć z podaną prędkością w prawo, to po sekundzie będzie on zajmować punkt [10, 0]). Przeprowadź symulację trwającą pełne 30 sekund. Gdy koła się zderzą, zostają unicestwione.

Ile kół zostanie po przeprowadzeniu symulacji? **[1,]** Zwizualizuj całą symulację w środowisku graficznym. **[2]**

---

[18] Algorytmów jest sporo, choćby algorytm Dijkstry. Jeżeli nawet nie znasz żadnego, to jest to czas na własne poszukiwania. Zadania-projekty zakładają poszukiwania i dokształcanie.

[19] Chodzi o algorytm służący do obliczenia wartości wielomianu w punkcie, oparty na przekształceniu klasycznej postaci wielomianu: $a^n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x^1 + a_0$ na postać: $x(x(x\ldots x(x(a_n) + a_{n-1}) + a_{n-2}) + \ldots + a_1) + a_0$. Polega to na wyłączaniu x przed nawias. Dzięki temu zmniejszamy znacząco ilość operacji mnożenia na rzecz operacji dodawania. Algorytm najlepiej zaimplementować za pomocą rekurencji: $\text{Horner}(x, n, \{a_n, a_{n-1}, \ldots, a_0\}) = x(\text{Horner}(x, n-1, \{a_n, \ldots, a_1\})) + a_0$.

#5 „Nerd”

Uwaga! W rozwiązaniach tej kategorii zadań posługuję się biblioteką graficzną 2D: SFML w wersji 2.5. Środowiskiem programistycznym, w którym wszystko kompiluję, jest Microsoft Visual Studio 2022. Projekty mają włączony standard C++20. Biblioteka SFML jest zainstalowana za pomocą narzędzia Menedżer Pakietów NuGet. Okno z grafiką, którą tworzę w rozwiązaniach, ma rozmiar 1920×1080 (Full HD).

#5 Program.hpp wykorzystywany w różnych zadaniach z SFML-em (z grafiką 2D)

```
// Program.hpp
#pragma once // gwarancja załadowania jeden raz
#include <SFML/Graphics.hpp>
#include <functional>
#include <map>
#include <list>
#include <iostream>
#include <typeinfo>
#define WIDTH 1920
#define HEIGHT 1080
#define FRAMELIMIT 60
using namespace std;
using namespace sf;
class mWindow : public RenderWindow {
public:
  mWindow() : RenderWindow(VideoMode(WIDTH, HEIGHT), L"SFML") {
    this->setPosition(Vector2i(0, 0));
    this->setKeyRepeatEnabled(false);
    this->setVerticalSyncEnabled(true);
  };
};
class Baza {
public:
  bool skasuj = false;
  virtual void progresuj() = 0;
  virtual void rysuj() = 0;
};

class Program {
public:
  inline static shared_ptr<mWindow> W = make_shared<mWindow>();
  inline static Event event;
  inline static list<shared_ptr<Baza>> obiekty;
  sf::Font czcionka;
  View kamera; // niekiedy chcemy innej kamery niż domyślna
  bool pokazCzas = true;
private:
  bool pauza = false;
  unsigned long long int pauzaPoMilisekundach = 0;  // po ilu milisekundach Pauza
  unsigned long long int pauzaPoKlatkach = 0; // po ilu klatkach Pauza
  sf::Clock zegar;
  sf::Text uplywCzasu;
```

```
  map <string, function<bool(void)>> testy = {
    {
      "zamknięcie okna",
      [&]() {
          if (event.type == Event::Closed or
            event.type == Event::KeyPressed and event.key.code == Keyboard::Escape)
              W->close();
          return false;
      }
    }
  };
  list <function<void(void)>> progresy;
  list <function<void(void)>> rysowanie;
  list <function<void(void)>> progresy_po;
public:
  Program() {
    czcionka.loadFromFile("arial.ttf");
    uplywCzasu.setFont(czcionka);
    uplywCzasu.setCharacterSize(30);
    uplywCzasu.setFillColor(Color::Magenta);
    uplywCzasu.setOutlineColor(Color::Green);
    uplywCzasu.setOutlineThickness(1);
    uplywCzasu.setString("czas");
    uplywCzasu.setStyle(Text::Italic);
    kamera.setSize({ WIDTH,HEIGHT });
  }
  // główna metoda uruchamiająca pętlę rysującą kolejne klatki
  void start() {
    zegar.restart();
    while (W->isOpen())
    {
      testyWydarzen();
      W->clear(Color::White);

      if (!czyPauza())
        for (auto& o : obiekty) {
          o->progresuj();
        }
      if (!czyPauza()) progresja();

      for (auto& o : obiekty) {
        o->rysuj();
      }
      rysuj();
      rysujUplywCzasu();

      if (!czyPauza()) progresja_po();

      W->display();

      if (pauzaPoMilisekundach != 0 and zegar.getElapsedTime().asMilliseconds() >
      ↪pauzaPoMilisekundach)
        pauza = true;

      if (!pauza and pauzaPoKlatkach > 0) {
        pauzaPoKlatkach--;
        if (pauzaPoKlatkach == 0) pauza = true;
```

```
    }

    for (auto it = obiekty.begin(); it != obiekty.end();) {
      if ((*it)->skasuj) {
        it = obiekty.erase(it);
      }
      else it++;
    }
  }
}
// rysuje upływ czasu w oknie 2D
void rysujUplywCzasu() {
  uplywCzasu.setPosition(W->getView().getCenter().x + WIDTH / 3,
  ↪W->getView().getCenter().y - HEIGHT / 2 + 40);
  uplywCzasu.setString(to_string(zegar.getElapsedTime().asMilliseconds()) + " milisek.");
  if (pokazCzas) W->draw(uplywCzasu);
}
template <typename T>
// ustawia środowisko w tryb PAUZY (bool) lub pauzuje po upływie pewnego czasu w ms
void ustawPauze(T parametr) {
  if (is_same_v<T, bool>) pauza = parametr;
  else pauzaPoMilisekundach = parametr;
}
// po ilu wyświetlonych klatkach ma być włączona pauza
void ustawKlatki(unsigned long long int k) { pauzaPoKlatkach = k; }
// ile klatek zostało do pauzy
unsigned long long int klatkiDoKonca() { return pauzaPoKlatkach; }
// test, czy jest pauza
bool czyPauza() { return pauza; }
// dodawanie funkcji wykonujących progres obiektów (zmiany stanów, np. ruch, pozycja, wielkości)
void dodaj_progres(function<void(void)>&& funkcja) {
  progresy.push_back(funkcja);
}
// dodawanie funkcji wykonujących progres obiektów (zmiany stanów, np. ruch, pozycja, wielkości)
void dodaj_progres_po(function<void(void)>&& funkcja) {
  progresy_po.push_back(funkcja);
}
// dodawanie funkcji rysujących
void dodaj_rysowanie(function<void(void)>&& funkcja) {
  rysowanie.push_back(funkcja);
}
// dodawanie funkcji związanych z obsługą zdarzeń
void dodaj_test(string klucz, function<bool(void)>&& funkcja) {
  testy[klucz] = funkcja;
}
// uruchamianie funkcji rysowania
void rysuj() {
  for (auto& f : rysowanie) f();
}
// uruchamianie funkcji progresji
void progresja() {
  for (auto& f : progresy) f();
}
// uruchamianie funkcji progresji
void progresja_po() {
  for (auto& f : progresy_po) f();
}
```

```
  // przechwytywanie i obsługa wydarzeń w oknie
  void testyWydarzen() {
    while (Program::W->pollEvent(Program::event)) {
      for (auto it = testy.begin(); it != testy.end(); ) {
        if (it->second()) it = testy.erase(it);
        else it++;
      }
      // pauza po wciśnięciu p
      if (Program::event.type == Event::KeyPressed and event.key.code == Keyboard::P)
      ↪pauza = !pauza;
    }
  }
};
inline Program program;
```

#5.1

```
// main.cpp
#include "Program.hpp"
#include <iostream>
#include <ctime>
#include <algorithm>
#define SLEEP 300
using namespace std;
////////////////////////////////////////////////////
class Kolo : public Baza, public sf::CircleShape {
public:
  Color org;
  Kolo(float f) : CircleShape(f) {
    org = Color(rand() % 256, rand() % 256, rand() % 256);
    setFillColor(org);
  }
  float r() { return getRadius(); }
  bool operator>(Kolo& inny) {
    return (this->r() > inny.r());
  }
  void zgas() {
    setFillColor(Color(org));
    setOutlineColor(Color::Transparent);
    setOutlineThickness(0);
    setPosition(getPosition().x, 300);
  }
  void zapal() {
    setFillColor(Color(255, 255, 255));
    setOutlineColor(Color(255, 0, 0));
    setOutlineThickness(5);
    setPosition(getPosition().x, 240);
  }
  void progresuj() { }

  void rysuj() {
    Program::W->draw(*this);
  }
};

class SortowanieBabelkowe {
public:
```

```
  size_t lewy=0;
  size_t ostatni=0;
  bool zmiana = false;
  vector<shared_ptr<Kolo>> do_posortowania;
  SortowanieBabelkowe() {
    vector<float> promienie;
    promienie.reserve(15);
    for (auto p = 10; p <= 100; p += 10) promienie.push_back(p);
    auto ile = rand() % 11 + 10;
    cout << "Sortujemy " << ile << " kółek.\n";
    float xpoz = 100;
    for (auto k = 1; k <= ile; k++) {
      auto r = promienie[rand() % promienie.size()];
      do_posortowania.push_back(make_shared<Kolo>(Kolo(r)));
      do_posortowania.back()->setOrigin(r, r);
      do_posortowania.back()->setPosition(xpoz, 300);
      Program::obiekty.push_back(do_posortowania.back()); // to się będzie rysowało
      xpoz += 100;
    }
    ostatni = do_posortowania.size() - 1;
    program.dodaj_progres(
      [&]() {
        this->krok();
      });
  }
  void krok() {
    for (auto& k : do_posortowania) k->zgas(); // zgaś wszystkie
    if (ostatni == 0) return;
    if (lewy == ostatni) {
      lewy = 0;
      ostatni--;
      sf::sleep(sf::milliseconds(SLEEP));
    }

    auto& k1 = do_posortowania[lewy];
    auto& k2 = do_posortowania[lewy + 1];
    k1->zapal();
    k2->zapal();
    if (lewy < ostatni) {
      if (*k1 > *k2) {
        if (!zmiana) {
          zmiana = true;
          sf::sleep(sf::milliseconds(SLEEP));
          return;
        }
        else zmiana = false;
        auto poz = k1->getPosition();
        k1->setPosition(k2->getPosition());
        k2->setPosition(poz);
        swap(k1, k2);
      }
      lewy++;
    }
    sf::sleep(sf::milliseconds(SLEEP/2));
  }
};
```

```
int main() {
  setlocale(LC_ALL, "");
  srand(time(0));
  SortowanieBabelkowe S;
  program.start();
}
```

Przed sortowaniem:

Po sortowaniu:

Pamiętaj, że grafika jest przykładowa, gdyż za każdym razem promienie kół mogą być różne.

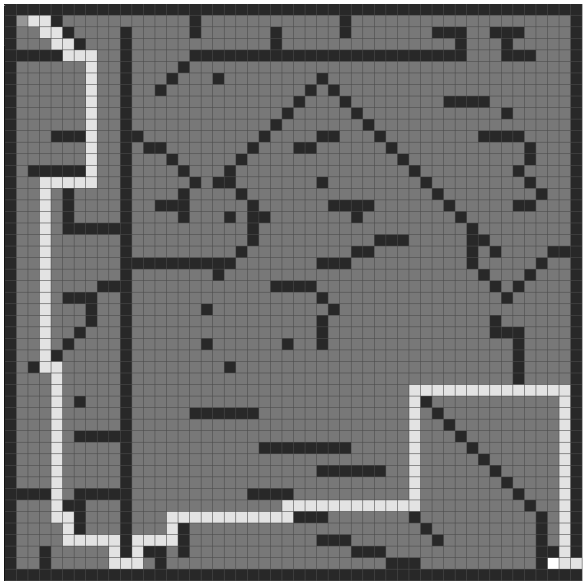
#5.2

```
// main.cpp
#include "Program.hpp" // dostępny na początku rozwiązań do części #5
#include <iostream>
#include <ctime>
#include <algorithm>
#include <fstream>
#include <queue>
#include <list>
constexpr float X = 19.;
constexpr float RAMKA = 1.;
using namespace std;
//////////////////////////////////////////////////////
enum class RODZAJ { ROCK = int('#'), VOID = int('.'), START = int('S'), END = int('K') };
class Pole : public Baza {
public:
  inline static unsigned long long int NR = 1;
  unsigned long long int nr = 0;
  bool przekraczalny = false;
  bool odwiedzony = false;
  RectangleShape blok;
  list<shared_ptr<Pole>> wyjscia; // góra, lewo, dół, prawo
  RODZAJ rodz;
  unsigned long long int licznik = 0;

  Pole(RODZAJ r) : rodz(r) {
    nr = NR++;
    blok.setSize(Vector2f(X, X));
    blok.setOutlineColor(Color(255, 0, 0));
```

```
    blok.setOutlineThickness(RAMKA);
    if (r == RODZAJ::ROCK) {
      blok.setFillColor(Color(40, 40, 40));
    }
    else {
      przekraczalny = true;
      if (r == RODZAJ::VOID) {
        blok.setFillColor(Color(200, 200, 200));
      }
      else if (r == RODZAJ::START) {
        blok.setFillColor(Color(100, 255, 100));
      }
      else {
        blok.setFillColor(Color(255, 100, 100));
      }
    }
  }
  void odwiedzaj() {
    blok.setFillColor(Color(120, 120, 120));
  }
  void zapal() {
    blok.setFillColor(Color(255, 255, 255));
  }
  void badany() {
    blok.setFillColor(Color(255, 0, 0));
  }

  void rysuj() {
    program.W->draw(blok);
  }
  void progresuj() {}
};

class Plansza : public Baza {
public:
  vector<vector<shared_ptr<Pole>>> plansza;
  queue<shared_ptr<Pole>> kolejka_odw;
  shared_ptr<Pole> start; // zapamiętam początek
  bool stop = false;
  Plansza(string plik) {
    ifstream odczyt(plik);
    string linia;
    for (int wiersz = 0; odczyt >> linia; wiersz++) {
      plansza.push_back({});
      plansza.back().resize(linia.size());
      for (int kolumna = 0; auto & znak : linia) {
        plansza[wiersz][kolumna] = make_shared<Pole>(Pole(RODZAJ(znak)));
        plansza[wiersz][kolumna]->blok.setPosition(kolumna * (X + RAMKA), wiersz *
        ↪ (X + RAMKA));
        if (RODZAJ(znak) == RODZAJ::START) {
          start = plansza[wiersz][kolumna];
          plansza[wiersz][kolumna]->odwiedzaj();
          kolejka_odw.push(plansza[wiersz][kolumna]);
        }
        kolumna++;
      }
    }
```

```
  odczyt.close();
  // tworzę listę możliwych wyjść z każdego pola na inne pole
  int wiersz = 0;
  for (auto& w : plansza) {
    int wsize = plansza.size();
    int kolumna = 0;
    for (auto& pole : w) {
      int ksize = w.size();
      // czy górne wyjście?
      if (wiersz - 1 >= 0 and plansza[wiersz - 1][kolumna]->przekraczalny)
        pole->wyjscia.push_back(plansza[wiersz - 1][kolumna]);
      // czy w prawo?
      if (kolumna + 1 < ksize and plansza[wiersz][kolumna + 1]->przekraczalny)
        pole->wyjscia.push_back(plansza[wiersz][kolumna + 1]);
      // czy w dół?
      if (wiersz + 1 < wsize and plansza[wiersz + 1][kolumna]->przekraczalny)
        pole->wyjscia.push_back(plansza[wiersz + 1][kolumna]);
      // czy w lewo?
      if (kolumna - 1 >= 0 and plansza[wiersz][kolumna - 1]->przekraczalny)
        pole->wyjscia.push_back(plansza[wiersz][kolumna - 1]);
      kolumna++;
    }
    wiersz++;
  }
}
void rysuj() {
  for (int wiersz = 0; wiersz < plansza.size(); wiersz++) {
    for (int kolumna = 0; kolumna < plansza[wiersz].size(); kolumna++) {
      plansza[wiersz][kolumna]->rysuj();
    }
  }
}
void progresuj() {
  if (stop) return; // już niczego nie analizuje, droga znaleziona (bądź nie)
  if (kolejka_odw.size() and kolejka_odw.front()->rodz != RODZAJ::END) {
    szukanieDrogi();
  }
  else if (kolejka_odw.size() and kolejka_odw.front()->rodz == RODZAJ::END) {
    pokazDroge();
  }
}
// algorytm szukanieDrogi dla grafów (traktujemy Pole jak wierzchołek grafu)
void szukanieDrogi() {
  static bool badany = false;
  auto& zkolejki = kolejka_odw.front();
  zkolejki->badany();
  if (!badany) {
    zkolejki->badany(); badany = true;
    return;
  }
  else {
    badany = false;
    // sf::sleep(sf::milliseconds(250)); // usuń znaki // rozpoczynające komentarz, by zobaczyć
    // dokładniej analizę
  }
  zkolejki->odwiedzaj();
  for (auto& wyjscie : zkolejki->wyjscia) {
    if (wyjscie != nullptr and !wyjscie->odwiedzony) {
```

```
        wyjscie->zapal();
        wyjscie->odwiedzony = true;
        wyjscie->licznik = zkolejki->licznik + 1;
        kolejka_odw.push(wyjscie);
        wyjscie = nullptr;
      }
    }
    cout << endl;
    kolejka_odw.pop();
  }

  void pokazDroge() {
    static auto odKonca = kolejka_odw.front();
    odKonca->blok.setFillColor(Color::Yellow);
    if (odKonca->rodz != RODZAJ::START) {
      for (auto& p : odKonca->wyjscia) {
        if (p != nullptr and p->licznik == odKonca->licznik - 1) {
          odKonca = p;
        }
      }
    }
    else stop = true;
    start->blok.setFillColor(Color::Green);
  }
};
int main() {
  setlocale(LC_ALL, "");
  srand(time(0));
  program.obiekty.push_back(make_shared<Plansza>("2_labirynt.txt"));
  program.start();
}
```

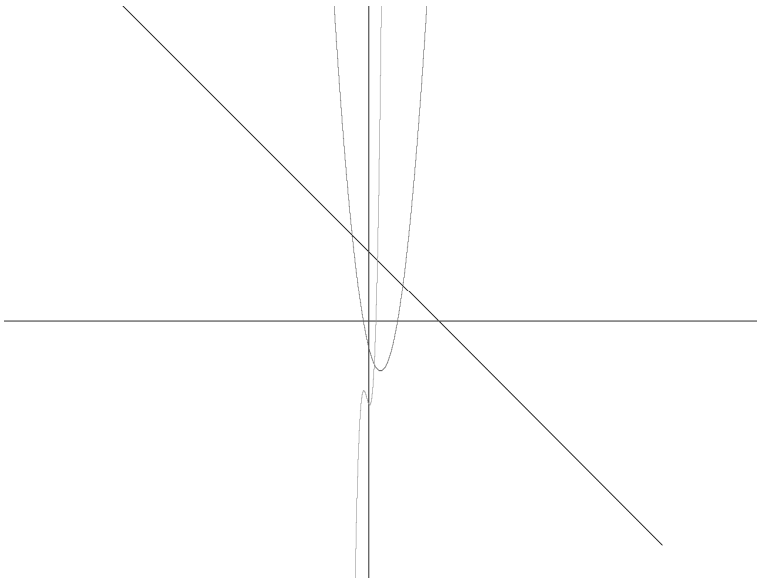


Na obrazku widać znalezioną drogę. W pliku *2_szukanie_drogi.mp4* będziesz mógł podejrzeć poszukiwanie.

#5.3

```
// main.cpp
#include "Program.hpp" // dostępny na początku rozwiązań do części #5
#include <iostream>
using namespace std;
//////////////////////////////////////////////////
double Horner(vector<double> wspolczynniki, double x) {
  if (wspolczynniki.size() == 1) return wspolczynniki.front();
  vector<double> kopia = wspolczynniki;
  auto ostatni = kopia.back();
  kopia.pop_back();
  return x * Horner(kopia, x) + ostatni;
}
class Wielomian : public Baza {
public:
  vector<double> wspolczynniki;
  double x0=0;
  double x1=0; // x0 : x1 -> zakres
  VertexArray va;
  RectangleShape osX;
  RectangleShape osY;
  Color c = Color::Black;
  Wielomian(vector<double>wsp,double zakres_x0, double zakres_x1) :
    wspolczynniki(wsp), x0(zakres_x0), x1(zakres_x1) {
    va.resize(abs(x1 - x0) + 1);
    va.setPrimitiveType(PrimitiveType::LineStrip);
    for (unsigned long long int poz = 0; poz < va.getVertexCount(); poz++) {
      va[poz].color = Color::Black;
      va[poz].position = Vector2f(x0,-Horner(wspolczynniki, x0));
      x0 += 1.;
    }
    osX.setFillColor(Color::Red);
    osX.setSize(Vector2f(WIDTH, 2));
    osX.setOrigin(Vector2f(WIDTH / 2, 1));
    osX.setPosition(0, 0);

    osY.setFillColor(Color::Red);
    osY.setSize(Vector2f(2,HEIGHT));
    osY.setOrigin(Vector2f(1,HEIGHT/2));
    osY.setPosition(0, 0);
  }
  auto&& ustawKolor(Color color) {
    c = color;
    for (auto poz = 0; poz < va.getVertexCount(); poz++) {
      va[poz].color = c;
    }
    return *this;
  }
  void rysuj() {
    program.W->draw(osX);
    program.W->draw(osY);
    program.W->draw(va);
    // dla pogrubienia va jego kopia "piksel niżej"
    auto vacopy = va;
    for (auto p = 0 ; p<vacopy.getVertexCount() ; p++ ) {
      vacopy[p].position = Vector2f(vacopy[p].position.x, vacopy[p].position.y + 1);
```

```
    }
    program.W->draw(vacopy);
  }
  void progresuj() {}
};
int main() {
  setlocale(LC_ALL, "");

  program.W->setView(View(Vector2f(0, 0),Vector2f(WIDTH,HEIGHT)));

  program.obiekty.push_back(make_shared<Wielomian>(
    Wielomian({ -1,120 }, -500, 500).ustawKolor(Color::Black)
  ));
  program.obiekty.push_back(make_shared<Wielomian>(
    Wielomian({ 0.1,-4,-44 }, -500, 500).ustawKolor(Color::Magenta)
  ));
  program.obiekty.push_back(make_shared<Wielomian>(
    Wielomian({ 0.05, .5, -2, -140 }, -500, 500).ustawKolor(Color::Green)
  ));

  program.start();
}
```



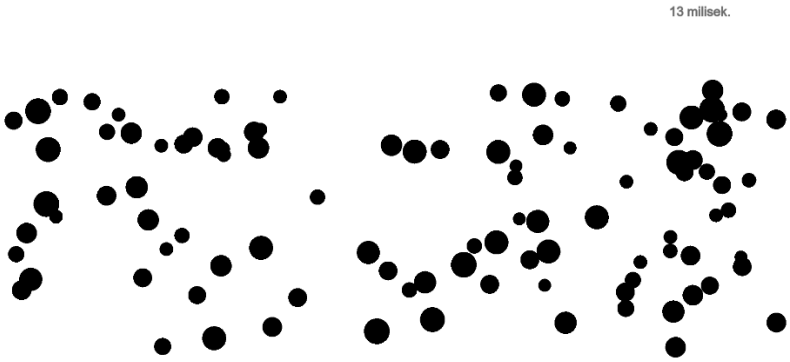
#5.4

```
// main.cpp
#include "Program.hpp" // dostępny na początku rozwiązań do części #5
#include <iostream>
#include <fstream>
#include <ctime>
using namespace std;
//////////////////////////////////////////////////////
class Kolo : public Baza {
public:
```

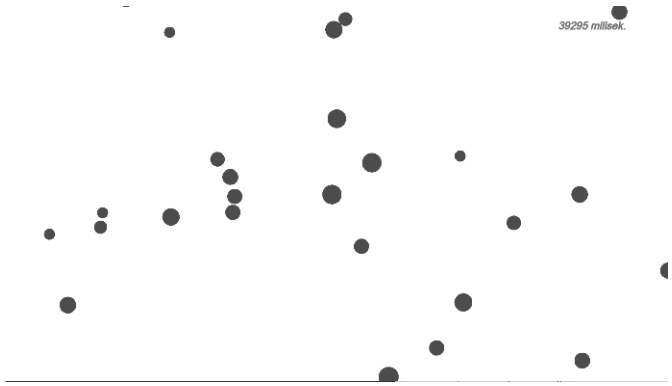
```
  CircleShape kolo;
  float kat;
  Kolo(float promien, float k) : kat(k) {
    kolo.setRadius(promien);
    kolo.setFillColor(Color::Black);
  }
  void rysuj() {
    if (program.klatkiDoKonca()==1) kolo.setFillColor(Color::Red);
    program.W->draw(kolo);
  }
  void progresuj() {
    float katRad = (3.14159265 / 180.) * (kat - 90);
    float celx = 1 * cos(katRad); // bo prędkość kół to 10/s (1 na 1/10 sekundy)
    float cely = 1 * -sin(katRad);
    kolo.move(celx, -cely);
  }

  // są 'równe', gdy kolidują
  bool operator==(Kolo& k) {
    // √((x_2-x_1)²+(y_2-y_1)²) (odległość dwóch punktów na płaszczyźnie)
    auto x2 = k.kolo.getPosition().x;
    auto x1 = kolo.getPosition().x;
    auto y2 = -k.kolo.getPosition().y;
    auto y1 = -kolo.getPosition().y;
    auto odleglosc = sqrt(abs((x2 - x1) * (x2 - x1) + (-y2 + y1) * (-y2 + y1)));
    return (odleglosc - (kolo.getRadius() + k.kolo.getRadius()) < 0.f);
  }
};
class Symulacja {
public:
  Symulacja() {
    ifstream dane("4_pozycje.txt");
    float x, y, promien, kat;
    while (dane >> x >> y >> promien >> kat) {
      auto k = make_shared<Kolo>(Kolo(promien, kat));
      k->kolo.setOrigin(promien, promien);
      k->kolo.setPosition(x, -y);
      program.obiekty.push_back(k);
    }
    cout << "Załadowano " << program.obiekty.size() << " obiektów.\n";
    dane.close();
  }
  void progresuj() {
    for (auto it = program.obiekty.begin(); it != program.obiekty.end(); it++) {
      for (auto it2 = program.obiekty.begin(); it2 != program.obiekty.end(); it2++) {
        if (it != it2) {
          auto k1 = dynamic_pointer_cast<Kolo>(*it);
          auto k2 = dynamic_pointer_cast<Kolo>(*it2);
          if (*k1 == *k2) { // kolizja
            k1->skasuj = k2->skasuj = true;
          }
        }
      }
    }
    cout << "Zostało obiektów: " << program.obiekty.size() << endl;
  }
};
```

```
int main() {
  setlocale(LC_ALL, "");
  srand(time(0));
  // ustaw kamerę na punkcie 0,0
  program.pokazCzas = false;
  program.W->setView(View(Vector2f(0, 0), Vector2f(WIDTH, HEIGHT)));
  // w każdej klatce koła ruszają się o 1, zatem przy prędkości 10/s
  // 10 klatek symuluje upływ czasu równy 1 sekundę
  program.ustawKlatki(10 * 30); // 300 klatek symuluje 30 sekund
  Symulacja S;
  program.dodaj_progres([&S]() { S.progresuj();  });
  program.start();
  // po wyjściu z okna graficznego zobaczymy, ile kół pozostało
  cout << "Zostało " << program.obiekty.size() << endl;
}
```

Przed uruchomieniem symulacji:



Po zatrzymaniu symulacji (niektóre koła są poza widokiem):



Zostało 27 obiektów (w zależności od dokładności symulacji dopuszczalny wynik to 25 kół).

C++ to język szerokiego zastosowania — od prostych programów konsolowych po zadania specjalne. Jest przenośny i niezależny od platformy, pozwala na samodzielne zarządzanie pamięcią, a przez kompilację do niskopoziomowego kodu zapewnia szereg optymalizacji i wysoką wydajność. Ta mieszanka wyjątkowych właściwości przekłada się na wszechstronność zastosowań C++ — to właśnie w nim powstały znane silniki gier, programy graficzne, przeglądarki, a nawet systemy operacyjne i oprogramowanie urządzeń IoT.

Znajomość C++ otwiera więc drzwi do kreowania cyfrowego świata w różnych dziedzinach. Jednakże początki z C++ mogą stanowić wyzwanie i wywoływać pytania, jak się uczyć, aby robić postępy, i w jaki sposób uporządkować zdobytą wiedzę. Cóż, nie ma lepszej metody niż nauka poprzez praktykę! Przed Tobą zbiór 100 zadań, skonstruowanych tak, by stopniowo zwiększać potrzebę stosowania coraz bardziej zaawansowanych elementów języka. Równocześnie wcześniejsze zadania są rozwiązywane takimi zasobami języka, jakie odpowiadają procesowi uczenia. Poza ćwiczeniami autor podzielił się licznymi ciekawostkami, sugestiami i uwagami skłaniającymi do przemyśleń i pogłębiania wiedzy o programowaniu.

Poświęcony C++ zbiór zaawansowanych zadań z rozwiązaniami:

- **jest podzielony na kategorie według sugerowanych przydatnych umiejętności i opatrzony opracowaną przez autora skalą trudności**
- **pomaga w usystematyzowaniu wiedzy**
- **pozwala wyrobić dobre praktyki dzielenia złożonych problemów na mniejsze części**
- **mobilizuje do łączenia zdobytych umiejętności w trakcie wykonywania ćwiczeń**
- **zawiera zadania z programowania grafiki 2D z użyciem biblioteki SFML**
- **obejmuje propozycje rozwiązań**

Te trzy starannie przygotowane rozdziały z zadaniami to o wiele więcej — to Twój osobisty motywator w nauce do matury, na studiach, a także w doskonaleniu kompetencji wymagających kreatywności i logicznego myślenia.

Przypływ wiedzy, niczym przypływ oceanu, podnosi jej poziom — fala za falą!

Kontynuacja książki ***C++. Zbiór zadań z rozwiązaniami***.

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej! ▶

ISBN 978-83-289-0181-0

9 788328 901810

Cena: 69,00 zł