Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э.Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э.Баумана)

Факультет «Робототехника и комплексная автоматизация»
Кафедра «Системы автоматизированного проектирования»

Отчет по технологической практике по теме: «Разработка программ изображения фигурных чисел»

Выполнил: студент группы РК6-44Б
Дунайцев А. И.
Научный руководитель: старший преподаватель кафедры РК6
Родионов С. В.

Дата: _____ Подпись: _____

Москва, 2021

Цель и задачи

Цель данной работы: Разработка программы для изображения фигурных чисел.
Задач работы:

1. Нахождение закономерностей для визуального изображения фигурных чисел
2. Разработка алгоритма для правильного изображения фигурного числа в общем случа
3. Программная реализация разработанного алгоритма
4. Составление вывод на основе проделанной работы

Общее описание работы

В рамках выполнения приведенных ранее задач будет разработанна программа на языке программирования общего назначения C++. Необходимо будет реализовать сущность, которая будет представлять собой фигурное число, содержать необходимые данные для вывода этого числа на экран и изобразить его в удобном для пользователя виде. Также для реализации данной программы необходимо будет создать класс оконного менеджера, который предоставляет интерфейс к графической системе ОС.

Программное и техническое обеспечение.
Программа выполняется на персональном компьютере под управлением операционной системы Ubuntu 20.04.02 LTS – дистрибутив Linux, основанный на Debian GNU/Linux.

Для написания программы использовался язык C++ - компилируемый, статически типизированный язык программирования общего назначения. Выбор этого языка обусловлен удобством его внутренней библиотеки, для которой реализованы необходимые структуры данных, а также существуют удобные и применимые для разработки графических приложений сторонние библиотеки.

Для написания кода использовалась интегрированная среда разработки для языков C и C++ CLion, разрабатываемая компанией JetBrains. CLion подходит для работы с выбранным языком программирование и подходит для операционной системы Ubuntu.

Для удобства компиляции и сборки кода использовалась система CMake и Makefile — это кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода. CMake не занимается непосредственно сборкой, а лишь генерирует файлы управления сборкой из файлов CMakeLists.txt: Makefile в системах Unix для сборки с помощью make. CMake хорошо подходит для задач сборки программ подобного рода, где необходимо линковать множество различных библиотек, как сторонних, так и написанных самостоятельно.

Библиотека SFML

SFML – свободная кроссплатформенная мультимедийная библиотека. Написана она на C++, но также доступна и на других языках программирования. SFML содержит ряд модулей для мультимедийного программирования приложений.

Главными преимуществами этой библиотеки является простота освоения, кроссплатформенность и, самое главное, быстрота.

SFML предоставляет следующие модули:

1. sfml-system
2. sfml-window
3. sfml-graphic
4. sfml-audio
5. sfml-network

Зависимости.
Для программ, которые будут запускаться на ПК под управлением Linux, SFML требует слудующие зависимости:

1. pthread
2. opengl
3. xlib
4. xrandr
5. udev
6. freetype
7. openal
8. flac
9. vorbis

SFML также имеет внутренние зависимости, аудио модуль и оконный модуль зависят от системного модуля, а графический модуль зависит от системного и оконного.

Установка.
Для того, чтобы установить SFML на ПК под управлением Linux, необходимо просто выполнить команду для пакетного менеджера apt и SFML сама установится из официального репозитория

```
sudo apt install libsfml-dev
```

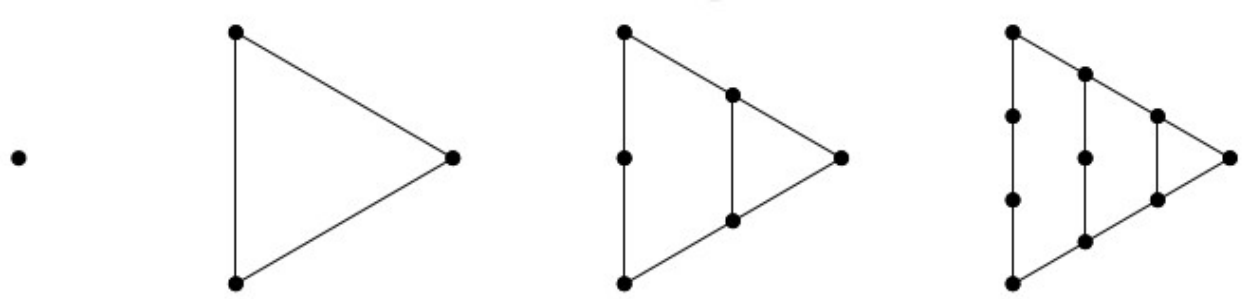
Описание алгоритма

Опираясь на теоретическую часть научно-исследовательской работы, следует отметить еще раз, что плоские многоугольные числа – положительные целые числа, соответствующие расположению точек на плоскости в виде правильного многоугольника.

Чтобы посторить многоугольное число, для начала необходимо взять точку на плоскости. Далее к этой точке необходимо добавить еще несколько точек таким образом, чтобы добавленая конфигурация точек представляла собой правильный многоугольник, например треугольник. Далее итерационно будем добавлять точки так, чтобы многоугольник оставался правильным, а количество его углов не менялось.

В итоге мы получим некоторую последовательность количства точек, которые мы добавляли на каждой итерации. Такие числа и их графические конфигурации назовем гномонами. Гномон – плоская фигура, при добавлении которой к исходной фигуре ее образует по форме такую же фигуру, но большего размера.

Так образуются числа в теории. На практике мы можем заметить, что на первую точку как бы наслаиваются гномоны, образуя многоугольники большего размрера.



Таким образом получается, что графически, можно отрисовывать набор правильных многоугольников, совмещая некоторые из точек так, чтобы они образовывали многоугольное число с правильным расположением точек на плоскости.

Следует отметить, что у каждого нового слоя есть набор из m углов для правильного m-угольника. Как раз одну из точек расположения заранее выбранного угла и будем совмещать с начальной.

Чтобы найти положение точек углов на плоскости воспользуемся формулой для вычисления центрального угла произвольного m-угольника $b = 360 / m$.

Теперь, мы можем найти на какой угол отклонилась та или иная точка, относительно некоторого начального положения, остается только выбрать расстояние. Зная эти параметры мы будем понимать положение необходимой точки в полярной системе координат, которые переведем в декартовы. Кроме того, расстояние до точки будет у каждого n-го m-угольного числа своим, но будет иметь общий, заранее условленный делитель, для того, чтобы можно было правильно сместить получившийся гномон. Нужно отметить, что

смещение на определенный отступ от начальной точки нужно делать для четных чисел в одну сторону, а для нечетных в другую, в зависимости от ориентации числа в пространстве.
Таким образом итерационно вычисляем точки для кажой из m-угольных конфигураций числа и вносим этот вектор точек в другую структуру данных, например еще один вектор, содержащий данные о всех доступных на данный момент конфигурациях.
Теперь основной задачей является необходимость расчитать количсество точек на каждом ребре числа.
Если обратить внимание на разные фигурные числа с разным количством углов, то можно заметить, что в общем случае, для n > 1, на кажой стороне m-угольника, будет n – 2 точки, не считая угловых.
Так как мы уже вычислили угловые, можно считать, что их координаты описывают грань, на которой должны располагаться промежуточные точки этой конфигурации. Теперь следует заметить, что эти точки делять грань на какое-то количество частей. Количество частей можно найти по формуле p = n – 1. Теперь, зная количество частей, на которые делится грань промежуточными точками, можем применить формулу по нахлждению координат точек, делящих грань в определенном отношении.

$$\lambda = \frac{AM}{BM} \quad x_M = \frac{x_A + \lambda \cdot x_B}{1 + \lambda}, \quad y_M = \frac{y_A + \lambda \cdot y_B}{1 + \lambda}$$

Теперь также итерационно найдем все промежуточные точки. Занесем найденные точки в множество всех точек.
Таким образом мы нашли координаты всех точек, которые необходимо расставить на плоскости, чтобы получился с виду правильный многоугольник, или набор правильных многоугольников, представляющих собой плоское многоугольное число.
Также программа, при помощи общей формулы для нахождения n-го m-угольного числа, на экран выводится число, характеризующее графическую конфигурацию на экране.
Следует отметить, что полученый алгоритм можно использовать для отрисовки центрированых многоугольных числе, разница заключается только в том, что в отличии от обычных многоугольных чисел, на не нужно сдвигать точки на некоторый отступ.
Остается только вывести это число на экран, используюя какие-либо средства позволяющие получить интерфейс к графической подсистеме ОС.
Программа будет представлена в листингах.

Выводы

В рамках выполнения данной технологической практики была написана программа, позволяющая изображать плоские фигурные числа. Программа обладает интерфейсом для выбора количества углов и порядковым номером числа. Также вся информация о числе выводится на экран.

Листинги программы

Файл PolygonalNumber.h.

```
#ifndef POLYGONALNUMBER_H
#define POLYGONALNUMBER_H

#include <vector>
#include <iostream>
#include <SFML/Graphics.hpp>

#define MIN_BASE 3
#define MIN_NUM 1
#define POINT_SIZE 4.0f
#define DEF_INDENT 20

#define CENTRAL_ANGLE(base) (360.0f / base)
#define GRAD_TO_RAD(grad) (grad * M_PI / 180.0f)

class PolygonalNumber {
public:
  PolygonalNumber(int base, int num, float indent);

  void draw(sf::RenderWindow &win);

  void SetPosition(float x, float y) {
    m_x = x;
    m_y = y;
  }

  void Increment() { appendGnomone(++m_num); }

  void Decrement() {
    if (m_num <= MIN_NUM) {
      return;
    }

    --m_num;
    points.pop_back();
  }

  int CalculateLineNumber() const;

  [[nodiscard]] int GetBase() const { return m_base; }
  [[nodiscard]] int GetNum() const { return m_num; }

private:
  struct Point {
    Point() : m_cond_x(0), m_cond_y(0) {}

    Point(float cond_x, float cond_y)
        : m_cond_x(cond_x), m_cond_y(cond_y) {
      point_body.setRadius(POINT_SIZE);
      point_body.setFillColor(sf::Color::Black);
    }

    sf::CircleShape point_body;

    float m_cond_x;
    float m_cond_y;
  };

  int m_base;
  int m_num;

  float m_central_angle;
  float m_x;
  float m_y;
  float m_indent;

  std::vector<std::vector<Point>> points;

  void appendGnomone(int n);
};


#endif //POLYGONALNUMBER_H
```

Файл PolygonalNumber.cpp.

```
#include <vector>
#include <math.h>

#include "PolygonalNumber.h"

PolygonalNumber::PolygonalNumber(int base, int num, float indent) :
      m_indent(indent), m_x(0), m_y(0) {
   if (base < MIN_BASE || num < MIN_NUM) {
      throw std::exception();
   }
   m_base = base;
   m_num = num;
   m_central_angle = CENTRAL_ANGLE(m_base);

   for (int i = MIN_NUM; i <= m_num; ++i) {
      appendGnomone(i);
   }
}

void PolygonalNumber::appendGnomone(int n) {
   std::vector<Point> gnomone;
   std::vector<std::pair<Point, Point>> edges(m_base);
   if (n == 1) {
      Point new_point(m_x, m_y);
      gnomone.emplace_back(new_point);
      points.push_back(gnomone);
      return;
   }

   float gnomone_center_indent = m_indent * ((float) n - 1);

   for (int i = 0; i < m_base; ++i) {
      float curr_angle = m_central_angle * (float) i;
      float curr_x = gnomone_center_indent * std::cos(GRAD_TO_RAD(curr_angle));
      curr_x += (m_base % 2 == 0 ? gnomone_center_indent : gnomone_center_indent * -1);
      float curr_y = gnomone_center_indent * std::sin(GRAD_TO_RAD(curr_angle));
      Point new_point(curr_x, curr_y);
      gnomone.push_back(new_point);

      edges[i].first = new_point;
      if (i == 0) {
         edges[edges.capacity() - 1].second = new_point;
      } else {
         edges[i - 1].second = new_point;
      }
   }

   if (n == 2) {
      points.push_back(gnomone);
      return;
   }

   int parts = n - 1;

   for (int i = 0; i < edges.size(); ++i) {
      Point &p1 = edges[i].first;
      Point &p2 = edges[i].second;

      for (int j = 1; j < parts; ++j) {
         float lambda = (float) j / ((float) parts - (float) j);
         float intermediate_x = (p1.m_cond_x + lambda * p2.m_cond_x) / (1 + lambda);
         float intermediate_y = (p1.m_cond_y + lambda * p2.m_cond_y) / (1 + lambda);

         Point intermediate_point(intermediate_x, intermediate_y);
         gnomone.push_back(intermediate_point);
      }
   }

   points.push_back(gnomone);
}

void PolygonalNumber::draw(sf::RenderWindow &win) {
   float is_odd = (m_base % 2 == 0 ? -1.0f : 1.0f);
   float addition_x = m_x + m_indent * m_num * is_odd;
   float addition_y = m_y;

   for (std::vector<Point> &gnomone : points) {
```

```
      for (Point &point : gnomone) {
        point.point_body.setPosition(sf::Vector2f(point.m_cond_x + addition_x, point.m_cond_y + addition_y));
        win.draw(point.point_body);
      }
    }
}

int PolygonalNumber::CalculateLineNumber() const {
    return (m_base * (m_num * m_num - m_num)) / 2 - m_num * m_num + 2 * m_num;
}
```

Файл WindowManager.h.

```
#ifndef WINDOWMANAGER_H
#define WINDOWMANAGER_H

#include <SFML/Graphics.hpp>

#include "PolygonalNumber.h"

#define WIN_TITLE "POLYGONAL NUMBERS"
#define FONT_SIZE 24

class WindowManager {
public:
    WindowManager() = delete;

    WindowManager(unsigned int win_width, unsigned int win_height, std::string font_file, PolygonalNumber &pn);

    void Dispatch();

private:
    sf::RenderWindow win;
    sf::Font font;
    sf::Text text;
    sf::Vector2u win_size;

    PolygonalNumber &polygonal_number;

    void draw();
    std::string getTextInfo();
    void keyboardHandler();
};

#endif //WINDOWMANAGER_H
```

Файл WindowManager.cpp.

```
#include <sstream>
#include <SFML/Graphics.hpp>

#include "WindowManager.h"
#include "PolygonalNumber.h"

WindowManager::WindowManager(unsigned int win_width, unsigned int win_height, std::string font_file, PolygonalNumber &pn)
      : win_size(sf::Vector2u(win_width, win_height)),
        win(sf::VideoMode(win_width, win_height), WIN_TITLE),
        polygonal_number(pn) {
    if (!font.loadFromFile(font_file)) {
        throw std::exception();
    }

    text.setFont(font);
    text.setCharacterSize(FONT_SIZE);
    text.setFillColor(sf::Color::Black);
}

void WindowManager::Dispatch() {
    win_size = win.getSize();
    polygonal_number.SetPosition((float)win_size.x / 2, (float)win_size.y / 2);

    while (win.isOpen()) {
        sf::Event event;
        while (win.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                win.close();
            }

            if (event.type == sf::Event::KeyPressed) {
```

```
            keyboardHandler();
        }

        draw();
      }
    }
}

void WindowManager::draw() {
    win.clear(sf::Color::White);

    std::string text_info = getTextInfo();
    text.setString(text_info);
    polygonal_number.draw(win);
    win.draw(text);
    win.display();
}

std::string WindowManager::getTextInfo() {
    std::stringstream info;
    info << "Base: " << polygonal_number.GetBase() << '\n'
    << "Number: " << polygonal_number.GetNum() << '\n'
    << "Equal to " << polygonal_number.CalculateLineNumber() << " line number";
    return info.str();
}

void WindowManager::keyboardHandler() {
    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
        polygonal_number.Increment();
    }

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
        polygonal_number.Decrement();
    }

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
        int curr_base = polygonal_number.GetBase();
        curr_base = (curr_base <= MIN_BASE ? MIN_BASE : --curr_base);
        int curr_num = polygonal_number.GetNum();
        polygonal_number = PolygonalNumber(curr_base, curr_num, DEF_INDENT);
        polygonal_number.SetPosition((float)win_size.x / 2, (float)win_size.y / 2);
    }

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
        int curr_base = polygonal_number.GetBase();
        int curr_num = polygonal_number.GetNum();
        polygonal_number = PolygonalNumber(++curr_base, curr_num, DEF_INDENT);
        polygonal_number.SetPosition((float)win_size.x / 2, (float)win_size.y / 2);
    }
}
```

Файл main.cpp.

```
#include "PolygonalNumber.h"
#include "WindowManager.h"

#define FONT_FILENAME "../project/static/font.ttf"

int main() {

    PolygonalNumber polygonalNumber(MIN_BASE, MIN_NUM, DEF_INDENT);
    WindowManager manager(800, 600, FONT_FILENAME, polygonalNumber);
    manager.Dispatch();

    return 0;

}
```