Отчет по Лабораторной работе №3

Технология программирования

Бекауов Артур Тимурович

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Написать компьютерную программу, содержащую описание классов для иерархии геометрических объектов (точка, линия, квадрат, ромб, прямоугольник, параллелограмм) с реализацией набора методов (изобразить, убрать, передвинуть, повернуть).

2 Ход лабораторной работы

Описание абстрактного класса grobject

Программа, написанная мной задаёт классы grobject, tline, tsquare, trekt, tromb, tpar.

Первым делом я создал окно вывода размером 1000 на 1000 пикселей, с заголовком `SFML`(Рис. 2.1). Я сделал это до описания всех классов, потому что почти все они используют окно в функции draw.

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include <cmath>

using namespace std;

sf::RenderWindow window(sf::VideoMode(1000, 1000), "SFML");
```

Рис. 2.1: Создание окна вывода

Затем я описал grobject - базовый, абстрактный класс (Рис. 2.2), поэтому экземпляров у него нет. Описывается четырьмя полями - начальная координата x `int x`, начальная координата y `int y`, цвет `int color` и видимость `int visible`.

```
//-----------------------------------------------------------
//----------Описание базового класса Точка-----------------
//-----------------------------------------------------------

//Переписываю базовый класс point в абстрактный класс grobject

class grobject
{
    protected:
        int x, y, color, visible;
        grobject();
        grobject(int xx,int yy);
        grobject(int xx, int yy, int c);
        ~grobject();

    public:
        virtual void draw() = 0; //чисто виртуальный метод draw - поэтому класс grobject абстрактный
        void hide();
        void reveal();
        void move(int dxx, int dyy);
};
```

Рис. 2.2: grobject: Описание класса

По существу класс grobject задаёт начальную точку цвет и видимость - параметры, которые унаследуют остальные классы.

Далее прописаны конструкторы и деструктор, вообще для абстрактного класса это не обязательно, но так как я переписывал абстрактный класс grobject из обычного класса point, у меня уже были конструкторы и деструктор, я решил их просто не убирать. Конструкторы и деструктор:

-grobject(); - конструктор по умолчанию (x=500, y=500, color=1, visible=1).

-grobject(int xx, int yy); - конструктор с заданными x, y (x=xx, y=yy, color=1,visible=1).

-grobject(int xx, int yy, int c); - конструктор с заданными x, y, color (x=xx, y=yy, color=c, visible=1).

-~grobject(); - деструктор.

После этого перечисленны методы абстрактного класса grobject (Рис. 2.3):

-virtual void draw() = 0; - чисто виртуальный метод draw - именно его наличие определяет класс grobject как абстрактный. В производных классах её придётся переопределять, в отличии от трёъ других функций, которые будут унаследованы всеми остальными классами.

-void hide(); - метод, который отключает видимость объекта, т.е он не будет отрисовываться на экране.

-void reveal(); - метод, который включает видимость объекта. т.е он будет отображаться на экране.

void move(dxx, dyy); - метод, который отклоняет координату начальной точки на dxx и dyy.

```
64  // Функция спрятать - выключает видимость
65  void grobject::hide()
66  {
67      visible = 0;
68  }
69
70  //Функция проявить - включает видимость
71  void grobject::reveal()
72  {
73      visible = 1;
74  }
75
76  // Функция Переместить - меняет коорд x,y.
77  void grobject::move(int dxx, int dyy)
78  {
79      x+=dxx;
80      y+=dyy;
81  }
82
```

Рис. 2.3: grobject: Методы

Описание производного (от grobject) класса tline

После этого я описал класс tline (производный от класса grobject) (Рис. 2.4), экземплярами которого являются прямые линии. Помимо полей описывающих начальную точку, видимость и цвет, унаследованных от класса grobject, класс tline имеет поля задающие конечную точку линии: отклонение конечной точки от начальной по координате x - `int dx` и отклонение конечной точки от начальной по координате y - `int dy`.

```
85  //-------------------------------------------------------------
86  //-----Описание производного (от точки) класса Линия--------
87  //-------------------------------------------------------------
88
89  class tline: public grobject
90  {
91      protected:
92          int dx, dy;
93
94      public:
95          tline();
96          tline(int xx, int yy, int dxx, int dyy);
97          tline(int xx, int yy, int dxx, int dyy, int c);
98          ~tline();
99
100         virtual void draw();
101         void rotate(double fi);
102 };
103
```

Рис. 2.4: tline: Описание класса

Далее прописаны конструкторы и деструктор класса tline

-tline(); - конструктор по умолчанию.

-tline(int xx, int yy, int dxx, int dyy); - конструктор с заданными параметрами x, y, dx, dy.

-tline(int xx, int yy, int dxx, int dyy, int c); - конструктор с заданными параметрами и цветом.

-~tline(); - деструктор.

Затем я прописал методы класса tline(Рис. 2.5 и Рис. 2.6). Помимо унаследованных от grobject методов hide, reveal и move, также были добавлены:

-virtual void draw(); - чисто виртуальный метод draw класса grobject был переопределён как виртуальный (фактический) метод draw класса tline. Если visible = 1, метод рисует линию в созданном в начале окне. Цвет линии определяется полем color (1 - красный, 2 - зелёный, 3 - синий, 4 - жёлтый, 5 - чёрный, ост - белый). (В этом классе и далее метод draw переопределяется как виртуальный. Сделано это для реализации механизма позднего связывания.)

-void rotate(double fi); - метод rotate. Вращает объект вокруг начальной (x,y) по часовой стрелке на угол fi `double fi`. Угол fi указывается в градусах.

```
147  // Функция вывода на экран
148  void tline::draw()
149  {
150      if (visible == 1)
151      {
152          sf::Vertex line[2];
153          line[0].position = sf::Vector2f(x,y);
154          line[1].position = sf::Vector2f(x+dx,y+dy);
155
156          switch(color)
157          {
158              case 1:
159              {
160                  line[0].color = sf::Color::Red;
161                  line[1].color = sf::Color::Red;
162                  break;
163              }
164              case 2:
165              {
166                  line[0].color = sf::Color::Green;
167                  line[1].color = sf::Color::Green;
168                  break;
169              }
170              case 3:
171              {
172                  line[0].color = sf::Color::Blue;
173                  line[1].color = sf::Color::Blue;
174                  break;
175              }
176              case 4:
177              {
178                  line[0].color = sf::Color::Yellow;
179                  line[1].color = sf::Color::Yellow;
180                  break;
181              }
182              case 5:
```

Рис. 2.5: tline: Методы (1\2)

```
            {
                line[0].color = sf::Color::Black;
                line[1].color = sf::Color::Black;
                break;
            }
            default:
            {
                line[0].color = sf::Color::White;
                line[1].color = sf::Color::White;
                break;
            }
        }
        window.draw(line, 2, sf::Lines);
    }
}

// Переписал - теперь прибавляет угол - крутит вокруг x,y по часовой стрелке - угол в градусах.
void tline::rotate(double fi)
{
    double PI = 3.14;
    double length = sqrt(dx*dx+dy*dy);

    int xi = int(asin(dy/length)*180/PI);

    dx=int(length*cos((xi+fi)*PI/180));
    dy=int(length*sin((xi+fi)*PI/180));
}

```

Рис. 2.6: tline: Методы (2\2)

Описание производного (от tline) класса tsquare

После этого я описал класс tsquare (производный от класса tline) (Рис. 2.7), экземплярами которого являются квадраты. Помимо полей описывающих начальную вершину, видимость, цвет и отклонение второй вершины от начальной, унаследованных от tline, класс tsquare не имее уникальных полей.

```
//-----------------------------------------------------------
//-----Описание производного (от линии) класса Квадрат------
//-----------------------------------------------------------

class tsquare: public tline
{
    public:
        tsquare();
        tsquare(int xx, int yx, int dxx, int dyy);
        tsquare(int xx, int yx, int dxx, int dyy, int c);
        ~tsquare();

        virtual void draw();
};

```

Рис. 2.7: tsquare: Описание класса

Далее прописаны конструкторы и деструктор класса tsquare:

-tsquare(); - конструктор по умолчанию.

-tsquare(int xx, int yy, int dxx, int dyy); - конструктор с заданными параметрами x, y, dx, dy.

-tsquare(int xx, int yy, int dxx, int dyy, int c); - конструктор с заданными параметрами и цветом.

-~tsquare(); - деструктор.

Затем я прописал методы класса tsquare (Рис. 2.8 и Рис. 2.9). Помимо унаследованных от tline методов hide, reveal, move, и rotate, также были добавлены:

–virtual void draw(); - был переопределён виртуальный метод draw. Если visible = 1, метод рисует квадрат в созданном в начале окне. Цвет квадрата определяется полем color.

```
270  void tsquare::draw()
271  {
272      if (visible == 1)
273      {
274          sf::VertexArray lines(sf::LineStrip,5);
275          lines[0].position = sf::Vector2f(x,y);
276          lines[1].position = sf::Vector2f(x+dx,y+dy);
277          lines[2].position = sf::Vector2f(x+dx-dy,y+dy+dx);
278          lines[3].position = sf::Vector2f(x-dy,y+dx);
279          lines[4].position = sf::Vector2f(x,y);
280
281
282          switch(color)
283          {
284              case 1:
285              {
286                  lines[0].color = sf::Color::Red;
287                  lines[1].color = sf::Color::Red;
288                  lines[2].color = sf::Color::Red;
289                  lines[3].color = sf::Color::Red;
290                  lines[4].color = sf::Color::Red;
291                  break;
292              }
293              case 2:
294              {
295                  lines[0].color = sf::Color::Green;
296                  lines[1].color = sf::Color::Green;
297                  lines[2].color = sf::Color::Green;
298                  lines[3].color = sf::Color::Green;
299                  lines[4].color = sf::Color::Green;
300                  break;
301              }
302              case 3:
303              {
304                  lines[0].color = sf::Color::Blue;
305                  lines[1].color = sf::Color::Blue;
306                  lines[2].color = sf::Color::Blue;
```

Рис. 2.8: tsquare: Методы (1\2)

```
307                 lines[3].color = sf::Color::Blue;
308                 lines[4].color = sf::Color::Blue;
309                 break;
310             }
311             case 4:
312             {
313                 lines[0].color = sf::Color::Yellow;
314                 lines[1].color = sf::Color::Yellow;
315                 lines[2].color = sf::Color::Yellow;
316                 lines[3].color = sf::Color::Yellow;
317                 lines[4].color = sf::Color::Yellow;
318                 break;
319             }
320             case 5:
321             {
322                 lines[0].color = sf::Color::Black;
323                 lines[1].color = sf::Color::Black;
324                 lines[2].color = sf::Color::Black;
325                 lines[3].color = sf::Color::Black;
326                 lines[4].color = sf::Color::Black;
327                 break;
328             }
329             default:
330             {
331                 lines[0].color = sf::Color::White;
332                 lines[1].color = sf::Color::White;
333                 lines[2].color = sf::Color::White;
334                 lines[3].color = sf::Color::White;
335                 lines[4].color = sf::Color::White;
336                 break;
337             }
338         }
339         window.draw(lines);
340     }
341 }
```

Рис. 2.9: tsquare: Методы (2\2)

Описание производного (от tsquare) класса trekt

После этого я описал класс trekt (производный от виртуального класса tsquare)(Рис. 2.10)(Класс tsquare в наследовании описан как виртуальный, равно как и у tromb. Сделано это потому что класс tpar будет иметь два предка (tromb и trekt), и унаследует два комплекта одинаковых полей tsquare от них. Чтобы избежать путанницы с уканием к какому полю идёт обращение, класс предок объявляется виртуальным, у trekt и tromb). Экземплярами класса являются прямоугольники. Помимо полей описывающих начальную вершину, видимость, цвет и отклонение второй вершины от начальной, унаследованных от tsquare, класс trekt имеет поля коэфициентов увеличения первой стороны `float ak` и второй стороны `float bk`. Первой стороной считается сторона между вершинами (x,y) и (x+dx,y+dy). Если ak и bk равны 1, то получается обычный квадрат с теми же параметрами.

```
348  //-------------------------------------------------------
349  //-Описание производного (от квадрата) класса прямоугольник-
350  //-------------------------------------------------------
351
352  class trekt: virtual public tsquare
353  {
354      protected:
355      float ak, bk; //Коэфы увел. сторон - 1 - без изм.
356
357      public:
358          trekt();
359          trekt(int xx, int yy, int dxx, int dyy, float akk, float bkk);
360          trekt(int xx, int yy, int dxx, int dyy, float akk, float bkk, int c);
361          ~trekt();
362
363          virtual void draw();
364  };
365
```

Рис. 2.10: trekt: Описание класса

Далее прописаны конструкторы и деструктор класса trekt:

-trekt(); - конструктор по умолчанию.

-trekt(int xx, int yy, int dxx, int dyy, float akk, float bkk); - конструктор с заданными параметрами x, y, dx, dy, ak, bk.

-trekt(int xx, int yy, int dxx, int dyy, float akk, float bkk, int c); - конструктор с заданными параметрами и цветом.

-~trekt(); - деструктор.

Затем я прописал методы класса trekt (Рис. 2.11 и Рис. 2.12). Помимо унаследованных от tsquare методов hide, reveal, move, и rotate, также были добавлены:

–virtual void draw(); - был переопределён виртуальный метод draw. Если visible = 1, метод рисует прямоугольник в созданном в начале окне. Цвет прямоугольника определяется полем color.

```
415  void trekt::draw()
416  {
417      if (visible == 1)
418      {
419          sf::VertexArray lines(sf::LineStrip,5);
420          lines[0].position = sf::Vector2f(x,y);
421          lines[1].position = sf::Vector2f(x+ak*dx,y+ak*dy);
422          lines[2].position = sf::Vector2f(x+ak*dx-bk*dy,y+ak*dy+bk*dx);
423          lines[3].position = sf::Vector2f(x-bk*dy,y+bk*dx);
424          lines[4].position = sf::Vector2f(x,y);
425
426
427          switch(color)
428          {
429              case 1:
430              {
431                  lines[0].color = sf::Color::Red;
432                  lines[1].color = sf::Color::Red;
433                  lines[2].color = sf::Color::Red;
434                  lines[3].color = sf::Color::Red;
435                  lines[4].color = sf::Color::Red;
436                  break;
437              }
438              case 2:
439              {
440                  lines[0].color = sf::Color::Green;
441                  lines[1].color = sf::Color::Green;
442                  lines[2].color = sf::Color::Green;
443                  lines[3].color = sf::Color::Green;
444                  lines[4].color = sf::Color::Green;
445                  break;
446              }
447              case 3:
448              {
449                  lines[0].color = sf::Color::Blue;
450                  lines[1].color = sf::Color::Blue;
451                  lines[2].color = sf::Color::Blue;
```

Рис. 2.11: trekt: Методы (1\2)

```
452                 lines[3].color = sf::Color::Blue;
453                 lines[4].color = sf::Color::Blue;
454                 break;
455             }
456             case 4:
457             {
458                 lines[0].color = sf::Color::Yellow;
459                 lines[1].color = sf::Color::Yellow;
460                 lines[2].color = sf::Color::Yellow;
461                 lines[3].color = sf::Color::Yellow;
462                 lines[4].color = sf::Color::Yellow;
463                 break;
464             }
465             case 5:
466             {
467                 lines[0].color = sf::Color::Black;
468                 lines[1].color = sf::Color::Black;
469                 lines[2].color = sf::Color::Black;
470                 lines[3].color = sf::Color::Black;
471                 lines[4].color = sf::Color::Black;
472                 break;
473             }
474             default:
475             {
476                 lines[0].color = sf::Color::White;
477                 lines[1].color = sf::Color::White;
478                 lines[2].color = sf::Color::White;
479                 lines[3].color = sf::Color::White;
480                 lines[4].color = sf::Color::White;
481                 break;
482             }
483         }
484         window.draw(lines);
485     }
486 }
```

Рис. 2.12: trekt: Методы (2\2

Описание производного (от tsquare) класса tromb

После этого я описал класс tromb (производный от виртульного класса tsquare) (Рис. 2.13), экземплярами которого являются ромбы. Помимо полей описывающих начальную вершину, видимость, цвет и отклонение второй вершины от начальной, унаследованных от tsquare, класс tromb имеет поле определяющее угол при второй вершине (x+dx,y+dy) `int fi`. Значение угла следует указывать в градусах, желательно от 0 до 180.

```
490  //-------------------------------------------------------
491  //-----Описание производного (от квадрата) класса ромб------
492  //-------------------------------------------------------
493
494  class tromb: virtual public tsquare
495  {
496      protected:
497      int fi; //значение угла между в вершине x+dx,y+dy в градусах (0;180)
498
499      public:
500          tromb();
501          tromb(int xx, int yy, int dxx, int dyy, int fik);
502          tromb(int xx, int yy, int dxx, int dyy, int fik, int c);
503          ~tromb();
504
505          virtual void draw();
506  };
507
```

Рис. 2.13: tromb: Описание класса

Далее прописаны конструкторы и деструктор класса tromb:

-tromb(); - конструктор по умолчанию.

-tromb(int xx, int yy, int dxx, int dyy, int fik); - конструктор с заданными параметрами x, y, dx, dy, fi.

-tromb(int xx, int yy, int dxx, int dyy, int fik, int c); - конструктор с заданными параметрами и цветом.

-~tromb(); - деструктор.

Затем я прописал методы класса tromb (Рис. 2.14 и Рис. 2.15 и Рис. 2.16). Помимо унаследованных от tsquare методов hide, reveal, move, и rotate, также были добавлены:

–virtual void draw(); - был переопределён виртуальный метод draw. Если visible = 1, метод рисует ромб в созданном в начале окне. Цвет прямоугольника определяется полем color.

```
558  void tromb::draw()
559  {
560      if (visible == 1)
561      {
562
563          double length = sqrt(dx*dx+dy*dy);
564          double PI = 3.14;
565
566          int xi = int(asin(dy/length)*180/PI);
567
568          int dxx = int((length*cos((xi+180-fi)*PI/180)));//можно поиграться с модулем
569          int dyy = int((length*sin((xi+180-fi)*PI/180))+1);
570
571          sf::VertexArray lines(sf::LineStrip,5);
572          lines[0].position = sf::Vector2f(x,y);
573          lines[1].position = sf::Vector2f(x+dx,y+dy);
574          lines[2].position = sf::Vector2f(x+dx+dxx,y+dy+dyy);
575          lines[3].position = sf::Vector2f(x+dxx,y+dyy);
576          lines[4].position = sf::Vector2f(x,y);
577
578
579          switch(color)
580          {
581              case 1:
582              {
583                  lines[0].color = sf::Color::Red;
584                  lines[1].color = sf::Color::Red;
585                  lines[2].color = sf::Color::Red;
586                  lines[3].color = sf::Color::Red;
587                  lines[4].color = sf::Color::Red;
588                  break;
589              }
590              case 2:
591              {
592                  lines[0].color = sf::Color::Green;
593                  lines[1].color = sf::Color::Green;
```

Рис. 2.14: tromb: Методы (1\3)

```
594                  lines[2].color = sf::Color::Green;
595                  lines[3].color = sf::Color::Green;
596                  lines[4].color = sf::Color::Green;
597                  break;
598              }
599              case 3:
600              {
601                  lines[0].color = sf::Color::Blue;
602                  lines[1].color = sf::Color::Blue;
603                  lines[2].color = sf::Color::Blue;
604                  lines[3].color = sf::Color::Blue;
605                  lines[4].color = sf::Color::Blue;
606                  break;
607              }
608              case 4:
609              {
610                  lines[0].color = sf::Color::Yellow;
611                  lines[1].color = sf::Color::Yellow;
612                  lines[2].color = sf::Color::Yellow;
613                  lines[3].color = sf::Color::Yellow;
614                  lines[4].color = sf::Color::Yellow;
615                  break;
616              }
```

Рис. 2.15: tromb: Методы (2\3)

```
617             case 5:
618             {
619                 lines[0].color = sf::Color::Black;
620                 lines[1].color = sf::Color::Black;
621                 lines[2].color = sf::Color::Black;
622                 lines[3].color = sf::Color::Black;
623                 lines[4].color = sf::Color::Black;
624                 break;
625             }
626             default:
627             {
628                 lines[0].color = sf::Color::White;
629                 lines[1].color = sf::Color::White;
630                 lines[2].color = sf::Color::White;
631                 lines[3].color = sf::Color::White;
632                 lines[4].color = sf::Color::White;
633                 break;
634             }
635         }
636         window.draw(lines);
637     }
638 }
```

Рис. 2.16: tromb: Методы (3\3)

Описание производного (от trekt и tromb) класса tpar

После этого я описал класс tpar (производный от классов tromb и trekt) (Рис. 2.17), экземплярами которого являются параллелограмы. Помимо полей описывающих начальную вершину, видимость, цвет , отклонение второй вершины от начальной, унаследованных от tsquare, класс tpar имеет поля коэфициентов увеличения сторон (унаследованные от trekt) и поле определяющее угол при второй вершине (унаследованное у tromb).

```
642 //-------------------------------------------------------
643 //---Производный (от ромба и прямоуг) класс параллелограмм--
644 //-------------------------------------------------------
645
646 class tpar: public tromb, public trekt
647 {
648  public:
649         tpar();
650         tpar(int xx, int yy, int dxx, int dyy, float akk, float bkk, int fik);
651         tpar(int xx, int yy, int dxx, int dyy, float akk, float bkk, int fik, int c);
652         ~tpar();
653
654         virtual void draw();
655 };
```

Рис. 2.17: tpar: Описание класса

Далее прописаны конструкторы и деструктор класса tpar:

-tpar(); - конструктор по умолчанию.

-tpar(int xx, int yy, int dxx, int dyy, float akk, float bkk, int fik); - конструктор с заданными параметрами x, y, dx, dy, ak, bk, fi.

-tpar(int xx, int yy, int dxx, int dyy, float akk, float bkk,int fik, int c); - конструктор с заданными параметрами и цветом.

-~tpar(); - деструктор.

Затем я прописал методы класса tpar (Рис. 2.18 и Рис. 2.19 и Рис. 2.20). Помимо унаследованных от tsquare методов hide, reveal, move, и rotate, также были добавлены:

–virtual void draw(); - был переопределён виртуальный метод draw. Если visible = 1, метод рисует параллелограм в созданном в начале окне. Цвет прямоугольника определяется полем color.

```
709  void tpar::draw()
710  {
711      if (visible == 1)
712      {
713          double length = sqrt(dx*dx+dy*dy);
714          double PI = 3.14;
715
716          int xi = int(asin(dy/length)*180/PI);
717
718          int dxx = int((length*cos((xi+180-fi)*PI/180)));//можно поиграться с модулем
719          int dyy = int((length*sin((xi+180-fi)*PI/180))+1);
720
721          sf::VertexArray lines(sf::LineStrip,5);
722          lines[0].position = sf::Vector2f(x,y);
723          lines[1].position = sf::Vector2f(x+ak*dx,y+ak*dy);
724          lines[2].position = sf::Vector2f(x+ak*dx+bk*dxx,y+ak*dy+bk*dyy);
725          lines[3].position = sf::Vector2f(x+bk*dxx,y+bk*dyy);
726          lines[4].position = sf::Vector2f(x,y);
727
728
729          switch(color)
730          {
731              case 1:
732              {
733                  lines[0].color = sf::Color::Red;
734                  lines[1].color = sf::Color::Red;
735                  lines[2].color = sf::Color::Red;
736                  lines[3].color = sf::Color::Red;
737                  lines[4].color = sf::Color::Red;
738                  break;
739              }
740              case 2:
741              {
742                  lines[0].color = sf::Color::Green;
743                  lines[1].color = sf::Color::Green;
```

Рис. 2.18: tpar: Методы (1\3)

```
744                 lines[2].color = sf::Color::Green;
745                 lines[3].color = sf::Color::Green;
746                 lines[4].color = sf::Color::Green;
747                 break;
748             }
749             case 3:
750             {
751                 lines[0].color = sf::Color::Blue;
752                 lines[1].color = sf::Color::Blue;
753                 lines[2].color = sf::Color::Blue;
754                 lines[3].color = sf::Color::Blue;
755                 lines[4].color = sf::Color::Blue;
756                 break;
757             }
758             case 4:
759             {
760                 lines[0].color = sf::Color::Yellow;
761                 lines[1].color = sf::Color::Yellow;
762                 lines[2].color = sf::Color::Yellow;
763                 lines[3].color = sf::Color::Yellow;
764                 lines[4].color = sf::Color::Yellow;
765                 break;
766             }
```

Рис. 2.19: tpar: Методы (2\3)

```
767             case 5:
768             {
769                 lines[0].color = sf::Color::Black;
770                 lines[1].color = sf::Color::Black;
771                 lines[2].color = sf::Color::Black;
772                 lines[3].color = sf::Color::Black;
773                 lines[4].color = sf::Color::Black;
774                 break;
775             }
776             default:
777             {
778                 lines[0].color = sf::Color::White;
779                 lines[1].color = sf::Color::White;
780                 lines[2].color = sf::Color::White;
781                 lines[3].color = sf::Color::White;
782                 lines[4].color = sf::Color::White;
783                 break;
784             }
785         }
786         window.draw(lines);
787     }
788 }
789
```

Рис. 2.20: tpar: Методы (3\3)

Функция int main()

В функции main я создал белую сетку из экземпляров класса tline, а также добавил по 3 экземпляра классов tsquare, tromb, trekt, tpar, один из которых в каждом классе скрываю функцией hide. (Рис. 2.21 и Рис 2.22).

После этого я описываю указатель на объект класса квадрат tsquare *fig. Затем я выделяю в динамической памяти объект, являющийся ромбом по умолчанию и присваиваю указателю fig значение этой памяти.

```
793
794 int main()
795 {
796
797     //Сетка
798     tline l4 = tline(0,200,1000,0,8);
799     tline l5 = tline(200,0,0,1000,8);
800     tline l6 = tline(0,400,1000,0,8);
801     tline l7 = tline(400,0,0,1000,8);
802     tline l8 = tline(0,600,1000,0,8);
803     tline l9 = tline(600,0,0,1000,8);
804     tline l10 = tline(0,800,1000,0,8);
805     tline l11 = tline(800,0,0,1000,8);
806
807     tsquare s1(600,600,100,100);
808     tsquare s2(600,200,100,100,2);
809     tsquare s3 = tsquare();
810     s1.hide();
811     s2.move(200,500);
812     s3.rotate(60);
813     s3.move(400,550);
814
815     trekt rk1 = trekt();
816     trekt rk2(100,100,100,100,2,1);
817     trekt rk3(800,100,200,0,0.5,2,4);
818     rk2.hide();
819     rk3.move(-50,410);
820     rk3.rotate(60);
821
```

Рис. 2.21: Функция main: Создание экземпляров (1\2)

```
824     tromb rm1 = tromb();
825     tromb rm2(300,500,300,100,120);
826     tromb rm3(300,100,200,100,150,3);
827     rm1.hide();
828     rm2.move(-260,-50);
829     rm2.rotate(-48);
830
831     tpar p1 = tpar();
832     tpar p2(200,100,200,100,3,1,60);
833     tpar p3(700,100,100,200,1,2,150,4);
834     p1.hide();
835     p2.move(0,-100);
836     p3.move(100,0);
837     p3.rotate(-3);
838
839     //Описание примера, демонстрирующего позднее связывание
840     tsquare *fig1;
841     fig1 = new tromb();
842
843
```

Рис. 2.22: Функция main: Создание экземпляров (2\2)

Далее идёт цикл, который выводит окно вывода на экран, в цикле описаны все выводимые на экран элементы (через функцию draw, ведь именно она отрисовывает объект на экране)(Рис. 2.23 и Рис 2.24).

```
846     while (window.isOpen())
847     {
848         sf::Event event;
849         while (window.pollEvent(event))
850         {
851             if (event.type == sf::Event::Closed)
852                 window.close();
853         }
854 
855         window.clear();
856 
857         //Сетка
858         l4.draw();
859         l5.draw();
860         l6.draw();
861         l7.draw();
862         l8.draw();
863         l9.draw();
864         l10.draw();
865         l11.draw();
866 
867         s1.draw();
868         s2.draw();
869         s3.draw();
```

Рис. 2.23: Функция main: Цикл вывода (1\2)

```
871         rk1.draw();
872         rk2.draw();
873         rk3.draw();
874 
875         rm1.draw();
876         rm2.draw();
877         rm3.draw();
878 
879         p1.draw();
880         p2.draw();
881         p3.draw();
882 
883         //Отрисовка примера демонстр. позднее связывание
884         fig1->draw();
885 
886         window.display();
887     }
888     return 0;
889 }
```

Рис. 2.24: Функция main: Цикл вывода (2\2)

Вывод программы следующий (Рис. 2.25):

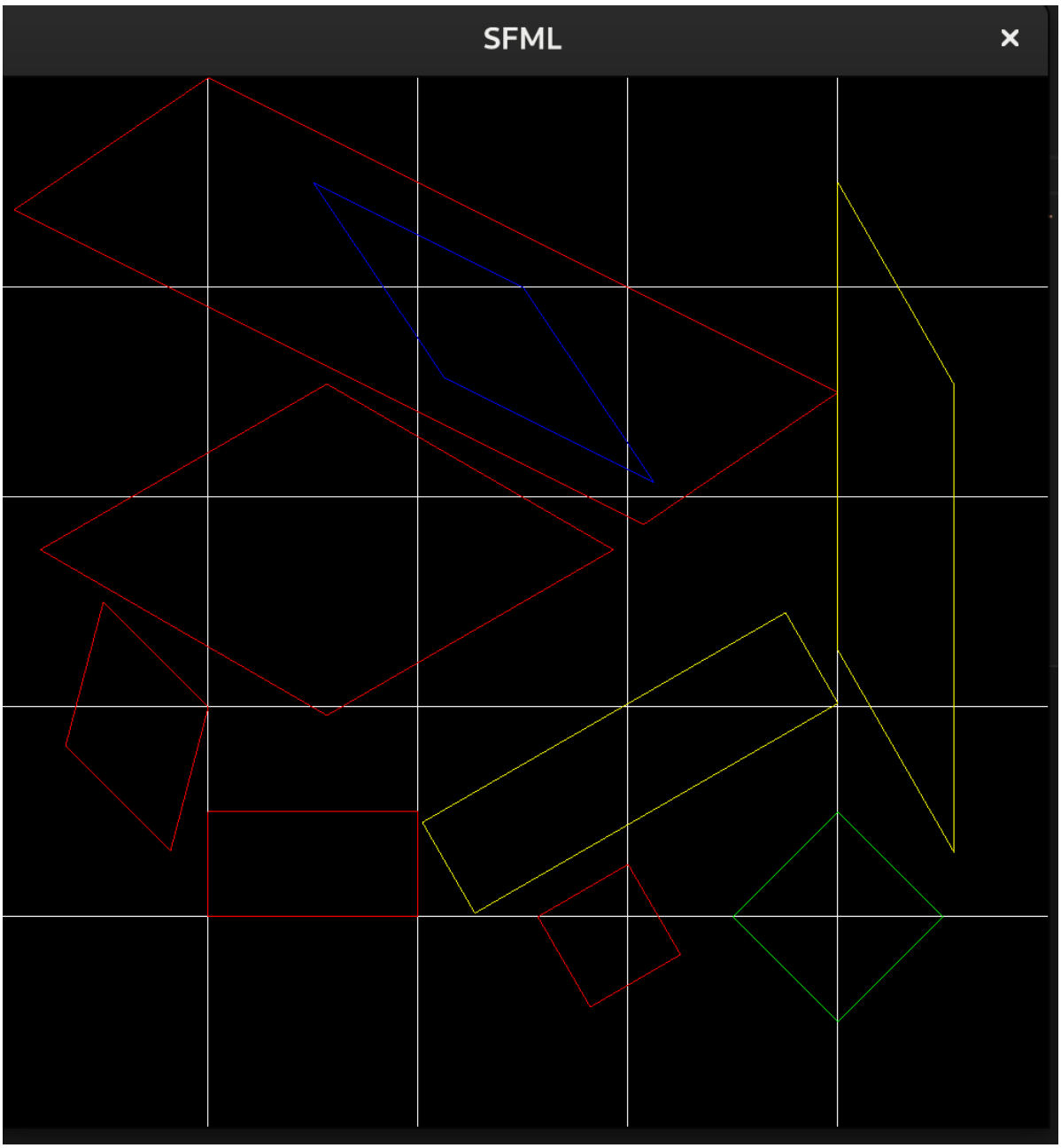

Рис. 2.25: Вывод программы

На рисунке отрисовывается третий ромб, значит, несмотря на то, что fig1 - указатель на экземпляр класса квадрат, на этапе позднего связывания программа понимает, что в динамической памяти находятся данные ромба и применяет соответствующий draw - из производного класса ромба.

3 Выводы

В ходе лабораторной работы я написал компьютерную программу на c++, содержащую описание классов для иерархии геометрических объектов (точка, линия, квадрат, ромб, прямоугольник, параллелограмм) с реализацией набора методов (изобразить, убрать, передвинуть, повернуть).