Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение
«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра прикладной математики и кибернетики

Расчетно-графическое задание по дисциплине «Объектно-ориентированное программирование». Задание №13: «Реализовать игру «Тетрис» в текстовом или в графическом режиме».

Выполнил:

студент 2 курса ИВТ

группы ИП – 811

Разумов Д. Б.

Проверила:

Ситняковская Е. И.

Новосибирск 2019

Содержание

Введение

Необходимый минимум содержания работы:

1. Инкапсуляция (все поля данных не доступны из внешних функций).
2. Наследование (минимум 3 класса, один из которых - абстрактный).
3. Полиморфизм.
4. Конструкторы, Перегрузка конструкторов.
5. Списки инициализации.

Я написал игру Тетрис с использованием объектно-ориентированных технологий: создан класс cell, который представляет собой клетку игрового поля, создан абстрактный класс figure, который служит родителем для классов фигур и использует класс cell. Проект программы состоит из четырех файлов: главного main.cpp, functions.h и functions.cpp для функций, classes.h для классов. Использована графическая библиотека graphics.h.

Некоторые понятия, которые я использовал в этом отчете: игровое поле - поле для игры в Тетрис, 20 клеток в высоту, 10 в ширину; клетки поля - это то, из чего поле состоит, клетки изначально имеют серый цвет, разделены они черными линиями; фигуры - классические фигуры из тетрамино, которые используются в Тетрисе и имеют отличные от серого и черного цвета.

Классы

```
 7  class cell {
 8  private:
 9      int x, y, color;
10      cell() : x(0), y(0), color(7) {}
11      cell (int x, int y, int c=7) { //параметр по умолчанию
12          this->x = x;
13          this->y = y;
14          color = c;
15      }
16      void draw(int dif) {
17          int pixel_y, pixel_x;
18          pixel_y = dif + y * dif + 1; //+1 для перемещения пера внуть клетки
19          pixel_x = dif + x * dif + 1; //
20          setfillstyle(1, color);
21          floodfill (pixel_x, pixel_y, 0); //заливка до черного цвета
22      }
23      void clear(int dif) {
24          int pixel_y, pixel_x;
25          pixel_y = dif + y * dif + 1; //+1 для перемещения пера внуть клетки
26          pixel_x = dif + x * dif + 1; //
27          setfillstyle(1, 7); //серый цвет
28          floodfill (pixel_x, pixel_y, 0); //заливка до черного цвета
29      }
30      void change_xy(int dx, int dy) {
31          x += dx;
32          y += dy;
33      }
34      friend class figure;
35      friend class T_figure;
36      friend class Z_figure;
37      friend class O_figure;
38      friend class I_figure;
39      friend class S_figure;
40      friend class J_figure;
41      friend class L_figure;
42  };
```

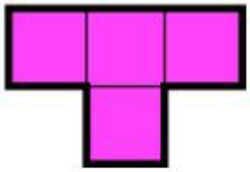
Как уже говорилось выше, класс cell представляет собой клетку игрового поля, поэтому он содержит координаты x и y, а также цвет (color) клетки. Есть перегрузка конструкторов. Первый конструктор содержит списки инициализации, второй конструктор содержит параметр по умолчанию. Метод draw() закрашивает клетку по координатам x, y на игровом поле в цвет из color. Метод clear() закрашивает клетку в изначальный серый цвет. Change_xy() меняет координаты клетки. Все поля и методы класса приватны, однако класс дружественен всем классам фигур, что позволяет полноценно использовать его в этих классах.

```
44  class figure {
45  private:
46      static const int n = 4;
47      virtual void position1() = 0;
48      virtual void position2() = 0;
49      virtual void position3() = 0;
50      virtual void position4() = 0;
51  protected:
52      cell cell_arr[n];
53      int posn_now, dif;
54  public:
55      change_posn() {
56          switch (posn_now) {
57              case 1:
58                  position2();
59                  break;
60              case 2:
61                  position3();
62                  break;
63              case 3:
64                  position4();
65                  break;
66              case 4:
67                  position1();
68                  break;
69              default:
70                  printf("\n error in switch \n");
71          }
72      }
73      void draw() {
74          for (int i=0; i<n; i++)
75              cell_arr[i].draw(dif);
76      }
77      void clear() {
78          for (int i=0; i<n; i++)
79              cell_arr[i].clear(dif);
80      }
81      int change_xy(int dx, int dy) {
82          int i, resul;
83          if (dx != 0) {
84              for (i=0; i<n; i++) {
85                  resul = check_cell(cell_arr[i].x + dx, cell_arr[i].y, dif);
86                  if (resul != 7)
87                      return 1;
88              }
89          } else
90          if (dy == 1) {
91              for (i=0; i<n; i++) {
92                  resul = check_cell(cell_arr[i].x, cell_arr[i].y + dy, dif);
93                  if (resul != 7)
94                      return 2;
95              }
96          }
97          for (int i=0; i<n; i++) {
98              cell_arr[i].change_xy(dx, dy);
99          }
100         return 0;
101     }
102     ~figure() {
103         //printf("destructor \n");
104     }
105 };
```

Класс figure является потомком для классов конкретных фигур. Так как каждая фигура в Тетрисе состоит из четырех клеток, я создал статическую переменную под это число. Виртуальные position1, position2 и другие меняют позицию фигуры (ее можно вращать на 90 градусов). Для каждой фигуры эти позиции реализуются по-разному, потому что каждая фигура уникальна. cell_arr[] - это массив объектов cell, из которых фигура состоит. posn_now - переменная для запоминания позиции. dif - длина и ширина клетки в пикселях. Методы draw(), clear() базируются на одноименных методах класса cell. Change_xy() выполняет ту же функцию, однако он так же проверяет (с помощью функции check_cell(), про которую будет рассказано дальше) правильность действий: нельзя перемещать фигуры за пределы игрового поля (возвращается значение 1). Также, если фигура достигла «дна» поля, то фигура перестает «падать» (возвращается 2).

```
106 class T_figure : public figure {
107 public:
108     T_figure(int dif) {
109         cell_arr[0] = cell(5, 0, 5);
110         cell_arr[1] = cell(4, 0, 5);
111         cell_arr[2] = cell(6, 0, 5);
112         cell_arr[3] = cell(5, 1, 5);
113         posn_now = 1;
114         this->dif = dif;
115     }
116     ~T_figure() {}
117 private:
118     void position1() {
119         if (check_cell(cell_arr[1].x-1, cell_arr[1].y-1, dif) == 7)
120         if (check_cell(cell_arr[2].x+1, cell_arr[2].y+1, dif) == 7)
121         if (check_cell(cell_arr[3].x-1, cell_arr[3].y+1, dif) == 7) {
122             cell_arr[1].change_xy(-1, -1);
123             cell_arr[2].change_xy(1, 1);
124             cell_arr[3].change_xy(-1, 1);
125             posn_now = 1;
126         }
127     }
128     void position2() {
138     void position3() {
148     void position4() {
158 };
```

T_figure - потомок figure. Он используется для данной фигуры, которая напоминает букву T.



Конструктор задает координаты используемых клеток и устанавливает начальную позицию. Далее реализуются позиции: сначала идет проверка на возможность поворота фигуры и только после этого меняются координаты клеток и posn_now. Методы position2, position3, position4 реализованы таким же способом. Другие классы фигур (Z, O, I, S, J, L) реализованы схожим образом с T-figure, несмотря на то, что у некоторых фигур существует только две или одна позиция.

```
212 class O_figure : public figure {
213 public:
214     O_figure(int dif) {
215         cell_arr[0] = cell(5, 0, 14);
216         cell_arr[1] = cell(6, 0, 14);
217         cell_arr[2] = cell(5, 1, 14);
218         cell_arr[3] = cell(6, 1, 14);
219         posn_now = 1;
220         this->dif = dif;
221     }
222     ~O_figure() {}
223 private:
224     void position1() {}
225     void position2() {}
226     void position3() {}
227     void position4() {}
228 };
```

Других классов нет.

Функции

```
3  int check_cell(int x, int y, int dif) {
4      int pixel_x = (x + 1) * dif + 1;
5      int pixel_y = (y + 1) * dif + 1;
6      return getpixel(pixel_x, pixel_y);
7  }
```

Данная функция принимает координаты и размер клетки, вычисляет координаты клетки в пикселях и возвращает цвет данной клетки. Эта функция используется в поворотах и передвижениях фигур, а также в другой функции.

```
8  void rand_array(int n, int arr[]) {
9      int i, u, temp;
10     for (i=0; i<n; i++) {
11         arr[i] = i;
12     }
13     for (i=0; i<n; i++) {
14         u = rand() % n;
15         temp = arr[i];
16         arr[i] = arr[u];
17         arr[u] = temp;
18     }
```

Данная функция принимает массив (который используется в main.cpp) для рандомизации появления фигур. В ней индексы (1, 2, 3...) фигур (T, Z, O…) перемешиваются между собой с помощью rand(). Можно определить появление фигур исключительно с помощью rand(), однако в таком случае часто бывает ситуация, в которой одна-две фигуры не появляются много раз подряд.

```
void move_lines(int main_y, int dif) {
    int x, y, pixel_x, pixel_y, color_top;
    for (y=main_y; y>0; y--) {
        pixel_y = (y + 1) * dif + 1;
        for (x=0; x<10; x++) {
            pixel_x = (x + 1) * dif + 1;
            color_top = getpixel(pixel_x, pixel_y - dif);
            setfillstyle(1, color_top);
            floodfill (pixel_x, pixel_y, 0); //заливка до черного цвета
        }
    }
    pixel_y = dif + 1;
    for (x=0; x<10; x++) { //самая верхняя строка
        setfillstyle(1, 7); //закрашивается в серый
        pixel_x = (x + 1) * dif + 1;
        floodfill (pixel_x, pixel_y, 0);
    }
}
```

Данная функция используется, когда одна или несколько линий заполнена фигурами - по правилам игра эта (эти) линии удаляются и все верхние линии смещаются вниз. Здесь запоминаются и используются цвета, для которой нужна функция check_cell().

```
void outtextxy(int x, int y, int val) { //вывод int числа
    char buf[20];
    snprintf(buf, sizeof(buf), "%d", val); //printf в массив buf
    outtextxy(x, y, buf);
}
```

Данная функция является перегруженной по отношению к outtextxy(int, int, *char) в библиотеке graphcs.h. С помощью функции snprintf() данные любого типа записываются в массив типа char. Этот массив используется в оригинальной outtextxt().

Алгоритм основной программы

```
1  #include "functions.h"
2  #include "classes.h"
3  #include <time.h>
4
5  int main() {
6      srand(time(0));
7      //--- графическое окно ----------------------------------------//
8      int x1, y1, dif, x2, y2, i, u;
9      x1 = y1 = dif = 40;
10     y2 = y1 + 20 * dif;
11     x2 = x1 + 10 * dif;
12     //x1, y1, x2, y2 - координаты поля для игры
13     int xmax=x2+dif*12, ymax=y2+dif;
14     initwindow(xmax, ymax);
```

Размер клетки равен 40 пикселям и записывается в переменную dif. x1, y1, x2, y2 - координаты поля для игры. Размеры поля: 20 на 10, поэтому x2, y2 вычисляются таким образом. Графическое окно смещено вправо на несколько сотен пикселей для отображения другой информации.

```
15     //--- создание игрового поля ---------------------------------//
16     Restart:
17     setfillstyle ( 1, 7 ); // стиль, цвет
18     bar (x1, y1, x2, y2);
19     setcolor(0);
20     int x, y, flag = 1, resul;
21     for (i=0, x=x1; i<10; i++, x+=dif) {
22         moveto(x, y1);
23         lineto(x, y2);
24     }
25     for (i=0, y=y1; i<20; i++, y+=dif) {
26         moveto(x1, y);
27         lineto(x2, y);
28     }
```

Restart - оператор для рестарта игры. Создается прямоугольник серого цвета, который потом за два цикла делится на клетки с помощью линий черного цвета.

```
29      //--- другая информация ------------------------------------//
30      setcolor(2);
31      settextstyle(0, 0, 3); //увеличение размера текста в три раза
32      outtextxy ( x2+dif, y2-240, "Управление:");
33      outtextxy ( x2+dif, y2-200, "W - вверх");
34      outtextxy ( x2+dif, y2-160, "A - влево");
35      outtextxy ( x2+dif, y2-120, "S - вниз");
36      outtextxy ( x2+dif, y2-80, "D - вправо");
37      outtextxy ( x2+dif, y2-40, "Enter - рестарт");
38      outtextxy ( x2+dif, y1, "Счет: ");
39      outtextxy ( x2+200, y1, 0); //перегрузка функции в functions.cpp
```

В данной части отображается информация об управлении и счете игрока - количество удаленных линий.

```
40      //--- массив фигур для избежания плохого рандома --------------//
41      int f_max = 7, f_i = 0, f_arr[f_max];
42      rand_array(f_max, f_arr);
43      figure *pfigure;
44      int key, score = 0;
45      while (flag) { //flag == 1, пока игра не закончена
46          //i = rand() % 6;
47          switch (f_arr[f_i]) {
48              case 0:
49                  pfigure = new Z_figure(dif);
50                  break;
51              case 1:
52                  pfigure = new T_figure(dif);
53                  break;
```

...

```
66              case 6:
67                  pfigure = new L_figure(dif);
68                  break;
69              default:
70                  break;
71          }
72          f_i++;
73          if (f_i == f_max) { //если все фигуры задействованы
74              f_i = 0;
75              rand_array(f_max, f_arr);
76          }
```

f_arr[] - это массив для индексов фигур, в rand_array() эти индексы перемешиваются. Далее запускается цикл, который заканчивается после того, когда нельзя поместить новую фигуру на игровом поле. С помощью switch() выбирается нужная фигура по индексу, здесь использован полиморфизм. Если все индексы использованы, то массив перемешивается заново.

```
77          pfigure->draw();
78          for (i=0; i<1000; i++) {
79              delay(5);
80              if ( kbhit() ) {
81                  key = getch();
82                  pfigure->clear();
83                  if (key == 75 || key == 97) //влево
84                      resul = pfigure->change_xy(-1, 0);
85                  if (key == 77 || key == 100) //вправо
86                      resul = pfigure->change_xy(1, 0);
87                  if (key == 80 || key == 115) //вниз
88                      resul = pfigure->change_xy(0, 1);
89                  if (key == 72 || key == 119) //вверх
90                      pfigure->change_posn();
91                  pfigure->draw();
92              }
93              if (i % 40 == 0) {
94                  pfigure->clear();
95                  resul = pfigure->change_xy(0, 1);
96                  pfigure->draw();
97              }
```

В цикле запускается другой цикл - он используется до тех пор, пока фигура не упадет вниз. С помощью методов draw() и clear() фигура рисуется или стирается. Если нажата (kbhit()) клавиши W, A, S или D, то соответсвенно координаты фигуры либо смещаются, либо она поворачивается на 90 градусов. Каждая 40-ая итерация принудительно смещает фигуру на одну клетку вниз.

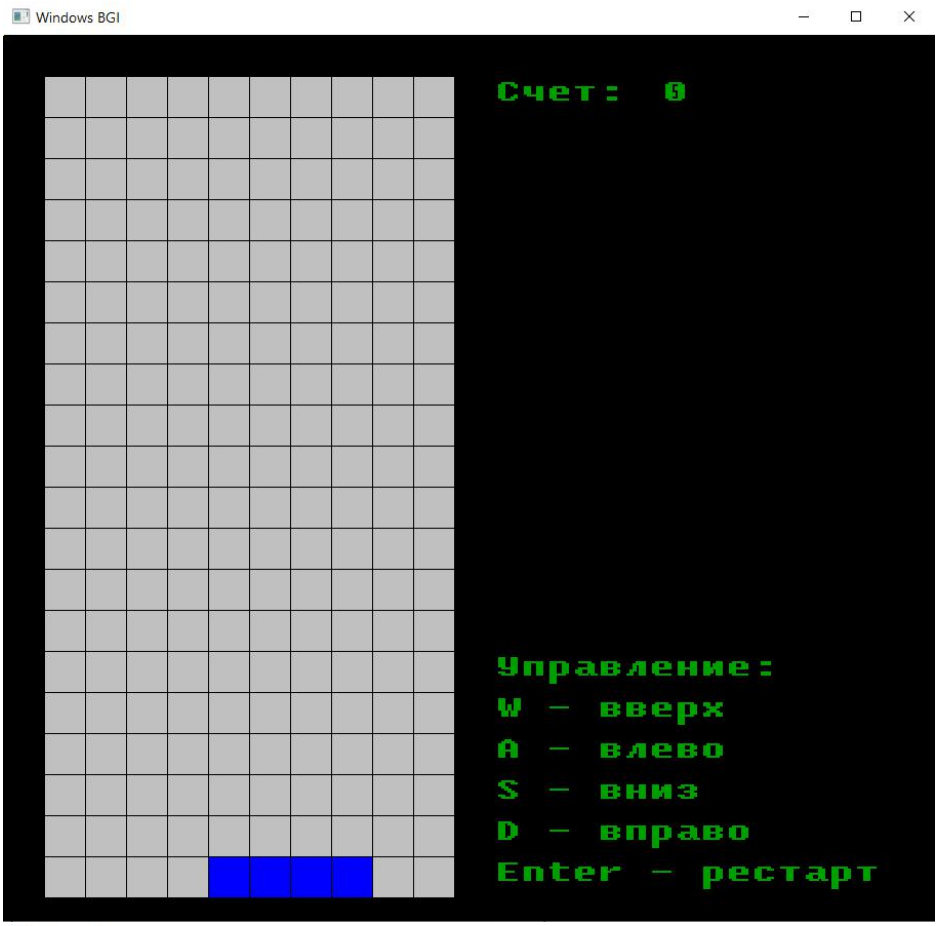
```
98              if (resul == 2) { //если фигура упала
99                  i = 1000;
100                 delete pfigure;
101             }
102             //проверка на завершение игры
103             if (i>40 and check_cell(5, 0, dif) != 7) {
104                 flag = 0;
105                 setcolor(4);
106                 settextstyle(0, 0, 5);
107                 outtextxy ( x2+dif, y1+50, "Game over");
108                 key = getch();
109                 if (key == 13) { //если enter, то рестарт
110                     setfillstyle ( 1, 0 ); //закрашивание надписи
111                     bar (x2+dif, y1, x2+500, y1+200);
112                     goto Restart;
113                 }
114             }
115         }
```

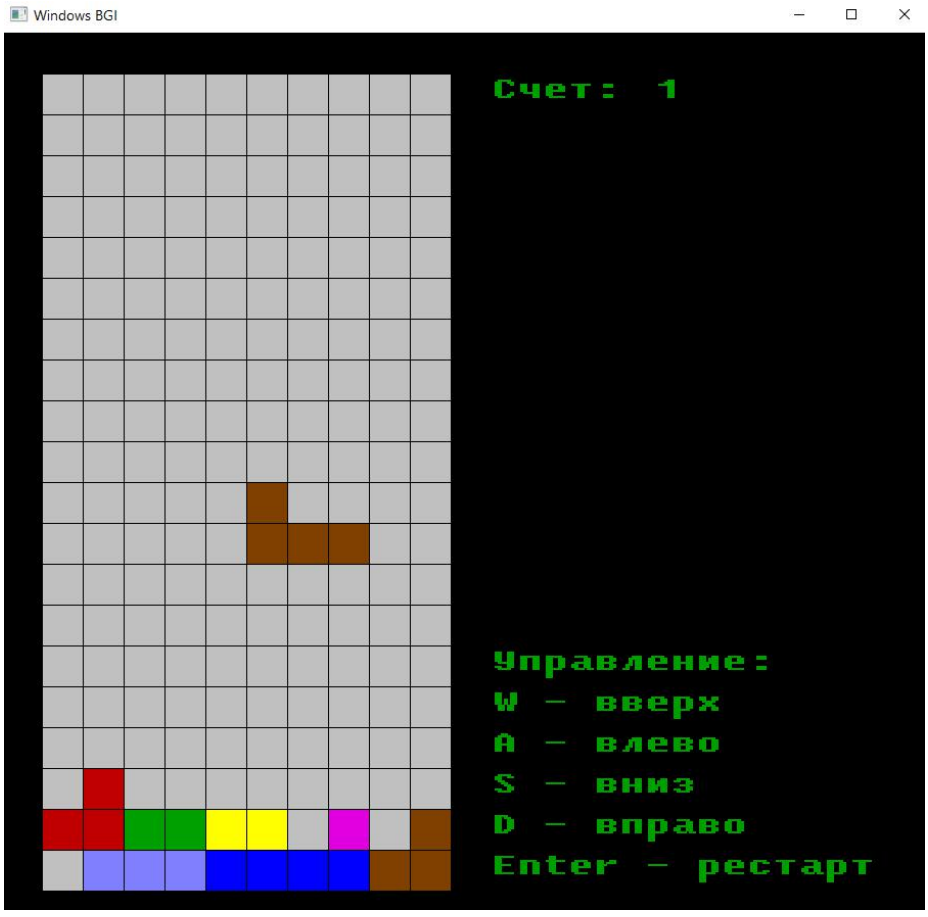
Как говорилось выше, если change_xy() возвращает 2, то это значит, что фигура упала на «дно» поля. Далле идет проверка на завершение игры. Координаты (5, 0) соответсвуют самой верхней клетке, в которой появляется любая из фигур. Здесь же можно начать игру заново, нажав на Enter.

```
116             //проверка на заполненные линии
117             for (i=0; i<20; i++) {
118                 y = (i + 1) * dif + 1;
119                 for (u=0, resul=0; u<10; u++) {
120                     x = (u + 1) * dif + 1;
121                     if (getpixel(x, y) != 7)
122                         resul++;
123                     else break;
124                 }
125                 if (resul == 10) { //если линия заполнена
126                     move_lines(i, dif);
127                     outtextxy ( x2+200, y1, ++score);
128                 }
129             }
130         }
131         getch();
132         return 0;
133 }
```

Далее идет проверка на заполненные линии. Берется цвет каждой клетки в линии, и если все 10 клеток не серого цвета, то запускается move_lines(). На этом цикл с flag и main.cpp заканчиваются.

Скриншоты


Windows BGI
Счет: 0
Управление:
W - вверх
A - влево
S - вниз
D - вправо
Enter - рестарт


Windows BGI
Счет: 1
Управление:
W - вверх
A - влево
S - вниз
D - вправо
Enter - рестарт