1. Задание

А снег идет, а снег идет
По массиву height[640] бьет, бьет…
ГлюкOZA C++. =)

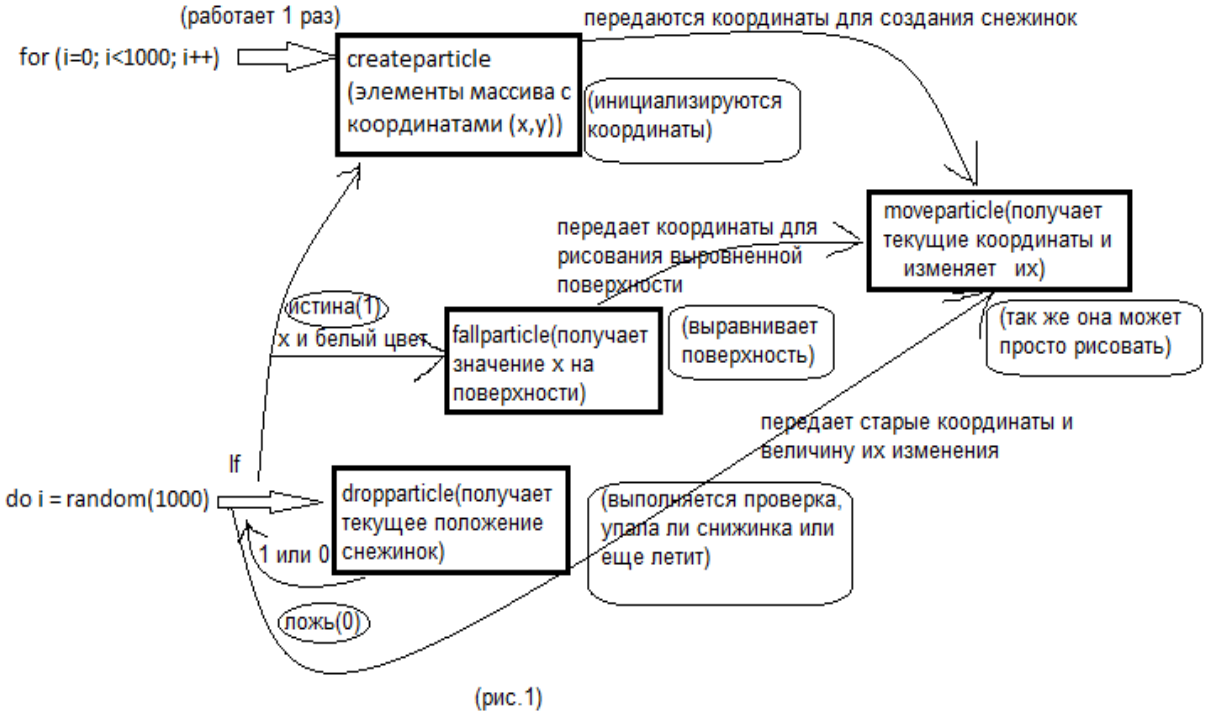
Нужно создать снегопад, где снежинки появляются из случайных точек и по всем поверхностям экрана образуются сугробы.

Есть проблема, что при падении на поверхность, сугробы будут «пикообразные», надо делать их ровными.

2. Структурное описание разработки

2.1. «Сложно о простом»

Снежинки представлены в виде элементов массива, а точнее их координаты (x,y)занесены в 2 массива. Несколько функций, описывающих их движение, получают координаты от других функций и возвращают все изменения в другие, то есть получается некоторый усложненный принцип вложенности. Подробную иллюстрацию можно увидеть в схеме (рис.1).



(рис.1)

Немного непонятно, но все таки эта схема описывает весь алгоритм. Получается, что все дороги ведут в Рим, точнее в «move». =)

2.2. «main – такой main»

Main начинается с создании поверхности. Поверхность представлена в виде массива height[640] (где 640 – размер экрана по x), задающегося в глобальных данных.

Заполняем его случайными значениями , и по этим значениям рисуем линии по вертикали.

```
for (int x = 0; x < 640; x++)
{
height[x] = random(100);
setcolor(1);
line(x,479,x,479-height[x]);
}
```

У нас получается неровная поверхность, похожая на скалы. Значит запускаем функцию выравнивания.

```
int change = 0;
do
  {
  change = 0;
  for (x = 0; x < 640; x++)
        {
        change |= fallparticle(x,9);
        }
  delay(10);
  }
while (change);
```

Она работает, пока полностью не выровняется поверхность.

Следующим шагом мы объявляем два массива:

```
int px[1000], py[1000];
```

Это будут наши ячейки, которые хранят координаты 1000 снежинок.

Запускаем цикл и инициализируем начальные координаты для самых первых снежинок, записывая их в массивы.

Запускаем бесконечный цикл. Задаем условие, что если результатом функции dropparticle() будет истина(1), то есть если снежинка упала, то на её место в массиве создается другая, а функция запускает еще одну функцию, которая «выравнивает» поверхность. Если ложь(0), то снежинка продолжает лететь.

```
do
  {
  i = random(1000);
  if (dropparticle(&(px[i]),&(py[i])))
        {
        createparticle(&(px[i]),&(py[i]));
        }
  }
while(!kbhit());
```

Вот так вкратце работает программа.

2.3. «Немного о прекрасном»

В процессе разработки программы, мне пришла идея, что пусть все будет в большей степени хаотично, случайно. Ведь, когда, во время снегопада, смотришь с высоты 13 этажа вниз, на дорогу, видишь, какая прекрасная суета в виде каши из людей и машин, стоящих в пробке из-за разбушевавшейся стихии.

Картинка стала образовываться сама собой. Кроме случайно образовывающихся снежинок в верхней части экрана, я решил сделать их движение хаотично и еще более того, что это хаотичное движение придавалось только случайно выбранным снежинкам. Каждый раз, запуская программу, мы видим разные картины этого снегопада. Вот одна из них (рис.2).

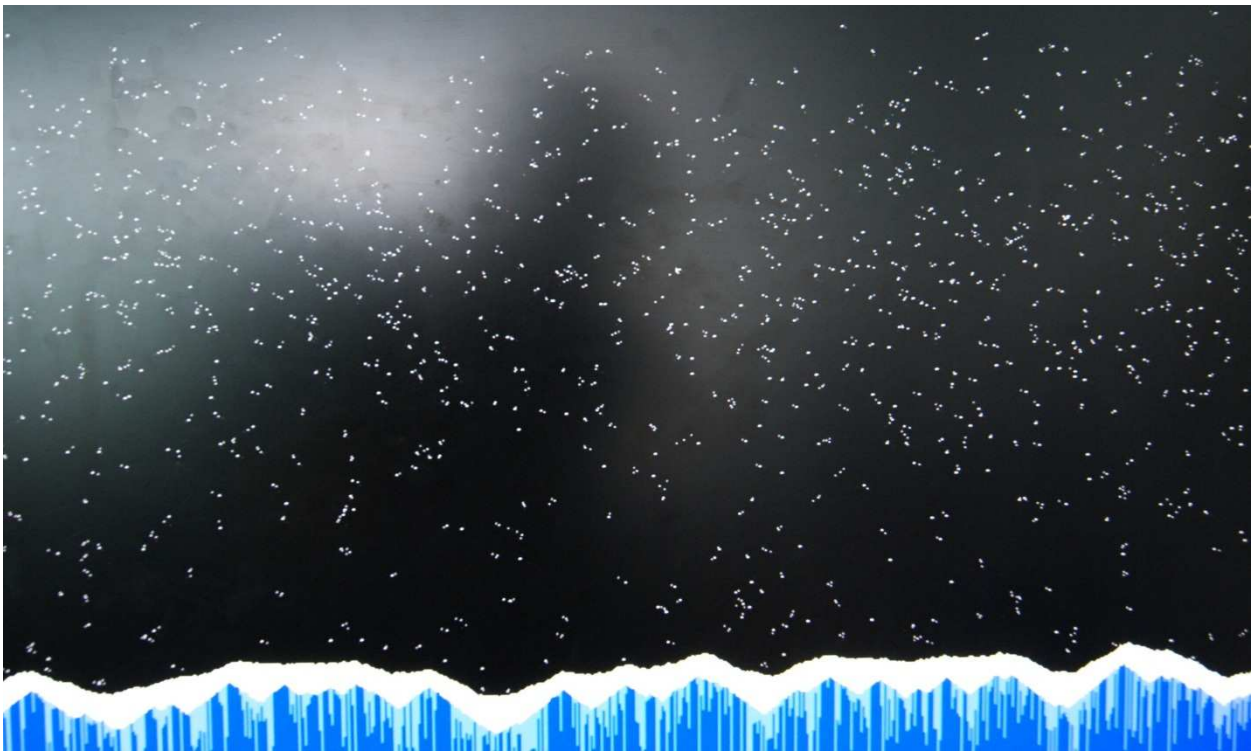


Рис.2

Здесь видно, что снежинки - пиксели экрана, а «сугробы» получаются не «пиками».

3. Функциональное описание

height[640] – глобальный массив.
void moveparticle() – функция рисования и движения путем закрашивания пикселей:
int:

x1-начальная координата x;
y1-начальная координата y;
x2-конечная координата x;
y2-конечная координата y;
c-цвет рисуемого пикселя.

int fallparticle() – функция выравнивающая поверхность. Работает, пока полностью не будут выполнятся условия.
Int:

x – текущая координата снежинки.
c – цвет.
change – переменная, которая фиксирует изменения.
result – переменная, которая сообщает main'у, что изменения произошли.

int dropparticle() – функция отвечающая, за передвижение снежинок, точнее проверяющая, не упала ли снежинка на поверхность.
Int:

*x – указатель на координату x.
*y – указатель на координату y.

xold – переменная, фиксирующая текущее положение снежинки по координате x.
yold – переменная, фиксирующая текущее положение снежинки по координате y.
void createparticle() – функция инициализирует начальные координаты снежинок.
Int:
*x – указатель на координату x.
*y – указатель на координату y.
void main()
int:
dr – драйвер.
mo – мод.
change – переменная, которая фиксирует, произошедшие изменения в fallparticle().
px[1000] – массив, хранящий координаты x 1000 снежинок.
py[1000] – массив, хранящий координаты y 1000 снежинок.
i – индекс элементов массива.

4. Описание работы программы

Ну что же, господа
присяжные заседатели,
начнем!
О.Бендер.

Запускаем нашу программу. В первую очередь запускается графический драйвер. Рисуется поверхность и тут же выравнивается.

```
for (int x = 0; x < 640; x++)
  {
  height[x] = random(100);
  setcolor(1);
  line(x,479,x,479-height[x]);
  }

int change = 0;
do
  {
  change = 0;
  for (x = 0; x < 640; x++)
    {
    change |= fallparticle(x,9);
    }
  delay(10);
  }
while (change);
```

Задаются 2 массива для хранения снежинок.

```
int px[1000],
py[1000];
```

Запускаем цикл с 1000 шагов, чтобы создать 1000 снежинок. Запускаем функцию creatparticle() для инициализации координат снежинок, а в самой функции запускается функция moverarticle(), чтобы их нарисовать.

```
for (int i = 0; i < 1000; i++)
  {
  createparticle(&(px[i]),&(py[i]));
  };
```

```
void moveparticle(int x1, int y1, int x2, int y2, int c)
{
putpixel(x1,479-y1,0);
putpixel(x2,479-y2,c);
}
```

После того, как нарисовались снежинки запускается бесконечный цикл do{ }while(!kbhit()). В нем выбираем случайным образом элемент массива, то есть ячейки, в которой хранятся координаты снежинки. Запускаем оператор условия, где условием будет являться результат функции dropparticle() 1 или 0. Если результатом будет 1, то будут рисоваться новые снежинки и цикл начнется заново.

```
do
  {
  i = random(1000);
  if (dropparticle(&(px[i]),&(py[i])))
    {
    createparticle(&(px[i]),&(py[i]));
    }
  }
while(!kbhit());
}
```

В самой функции dropparticle() проверяется, упала ли снежинка на поверхность или нет.

```
int dropparticle(int *x, int *y)
{
int xold = *x,
  yold = *y;
…..
…..
}
```

Если нет, то задается случайная длина отклонения и проверяется, не ушла ли снежинка с этими отклонениями за пределы экрана, если ушла, то длина отклонения корректируется. Если отклонение снежинки «попадет» на поверхность, то отклонение сбрасывается, что позволяет снежинки «нормально» коснуться поверхности. Отклонение передается функции moverarticle(), которая рисует движение снежинки. Результатом работы функции передается 0.

```
if (*y > height[*x])
  {
  (*y)--;
  (*x)+=random(5)-2;
  if ((*x) == 640)
  (*x)--;
  if ((*x) == -1)
  (*x)++;
  if (height[*x] >= *y)
  (*x) = xold;
  moveparticle(xold,yold,*x,*y,15);
  return(0);
  };
```

Если снежинка упала на поверхность, то к «столбику» поверхности добавляется один пиксель с помощью функции moveparticle() и сразу запускается функция fallparticle(), которая выравнивает поверхность относительно этой точки.

```
if (*y <= height[*x]+1)
```

```
{
height[*x]++;
moveparticle(*x,height[*x],*x,height[*x],15);
fallparticle(*x,15);
return(1);
}
```

В самой функции fallparticle() выполняются следующие действия: нам нужно сделать так, чтобы разница между соседними «столбиками» поверхности было минимальное (не больше 1). Нам нужно запустить 2 условия, что бы осуществить 2 проверки (слева и справа).
Значит для проверки справа, мы проверяем, не является ли столбик «крайним справа» и смотрим, больше ли текущий столбик соседа справа на >1, если да, то убираем один пиксель из текущего столбца и добавляем в соседний справа все тем же пресловутым moveparticle(), если нет, то ничего не делаем. Если изменения произошли, то переменной change присвоится 1.

```
if (x < 639)
    if (height[x+1] < (height[x]-1))
    {
    height[x+1]++;
    height[x]--;
    moveparticle(x,height[x]+1,x+1,height[x+1],c);
    x++;
    change = 1;
    }
```

Для проверки слева, мы проверяем, не является ли столбик «крайним слева» и не было ли изменений для проверки справа (если были, то ничего не происходит). Далее мы проверяем больше ли текущий столбик соседа слева на >1, если да, то убираем один пиксель из текущего столбца и добавляем в соседний слева. Так же переменной change присваивается 1.

```
if ((x > 0) && (!change))
        if (height[x-1] < (height[x]-1))     //Proverka sleva
        {
        height[x-1]++;
        height[x]--;
        moveparticle(x,height[x]+1,x-1,height[x-1],c);
        x--;
        change = 1;
        }
```

Результатом функции будет 1(если было хоть одно изменение) или 0 (если изменений не было).

Вот так, по принципу вложенности, работает данная программа.

Приложение. Исходный текст программы

```
#include <graphics.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>

int height[640]; //Массив поверхности

void moveparticle(int x1, int y1, int x2, int y2, int c) //Функция рисования
{
putpixel(x1,479-y1,0); //Удаляем старый
putpixel(x2,479-y2,c); //Рисуем новый
}

int fallparticle(int x, int c)  // Функция выравнивания
{
int change,
        result = 0;
do
  {
  change = 0;
  if (x < 639)
        if (height[x+1] < (height[x]-1)) //Проверка справа
        {
        height[x+1]++; //Прибавляем соседу
        height[x]--; //Убавляем текущему
        moveparticle(x,height[x]+1,x+1,height[x+1],c); //Рисуется прибавка/отбавка
        x++;
        change = 1; //Фиксируется изменение
        }
  if ((x > 0) && (!change))
        if (height[x-1] < (height[x]-1))     //Проверка слева
        {
        height[x-1]++; //Прибавляем соседу
        height[x]--; //Убавляем текущему
        moveparticle(x,height[x]+1,x-1,height[x-1],c); //Рисуется прибавка/отбавка
        x--;
        change = 1; //Фиксир. изменения
        }
  result |= change; //Фиксируется были ли изменения поверхности
  }
while (change);
return(result);
}

int dropparticle(int *x, int *y) //Функция проверки полета снежинки
{
int xold = *x,
        yold = *y;
if (*y > height[*x]) //Если летит
        {
        (*y)--; //опускается на 1 пкс.
        (*x)+=random(5)-2; //случайная длина отклонения
        if ((*x) == 640) //чтобы не вылетело за экран
        (*x)--;
        if ((*x) == -1) //чтобы не вылетело за экран
        (*x)++;
        if (height[*x] >= *y) //чтобы не попало на поверхность
        (*x) = xold;
        moveparticle(xold,yold,*x,*y,15); //рисуем движение
        return(0);
```

```
            };
if (*y <= height[*x]+1) //если упала
            {
            height[*x]++; //добавляем к столбику пкс.
            moveparticle(*x,height[*x],*x,height[*x],15); //Создаем снежинку
            fallparticle(*x,15); //выравниваем поверхность
            return(1);
            }
}

void createparticle(int *x,int *y) //создаем координаты снежинок
{
*x=random(640);
*y=459-random(50);
moveparticle(*x,*y,*x,*y,15); //рисуем снежинку
}

void main()
{

int dr = DETECT,mo;
initgraph(&dr,&mo,"e:\\bc31\\bgi"); //загрузка граф. драйвера

for (int x = 0; x < 640; x++)        //рисуем поверхность
            {
            height[x] = random(100);
            setcolor(1);
            line(x,479,x,479-height[x]);
            }

int change = 0; //Выравниваем
do
  {
  change = 0;
  for (x = 0; x < 640; x++)
              {
              change |= fallparticle(x,9);
              }
  delay(10);
  }
while (change);


int px[1000],  //для хранения
            py[1000];

for (int i = 0; i < 1000; i++) //создаем первую кучу снежинок
            {
            createparticle(&(px[i]),&(py[i]));
            };

do
  {
  i = random(1000); //случайно выбираем снежинку
  if (dropparticle(&(px[i]),&(py[i]))) //проверяем полет, если летит, то 0, если нет, то 1
              {
              createparticle(&(px[i]),&(py[i])); //если 1, то создаем новые координаты
              }
  }

while(!kbhit());

}
```