《计算机图形学》系统设计11月进展报告

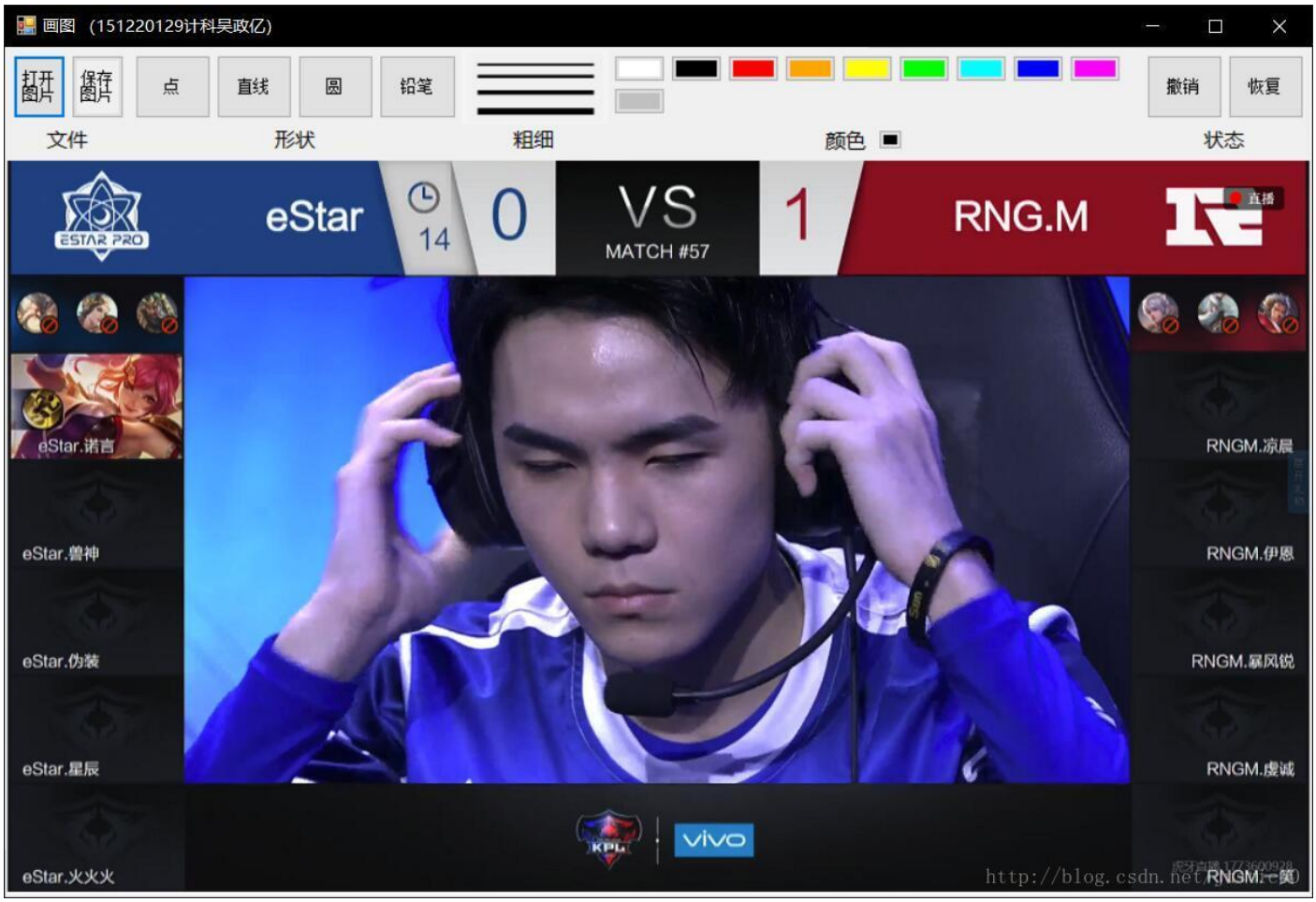
151220129 计科 吴政亿

（南京大学 计算机科学与技术系，南京 210093）

摘要

十月月更新进度

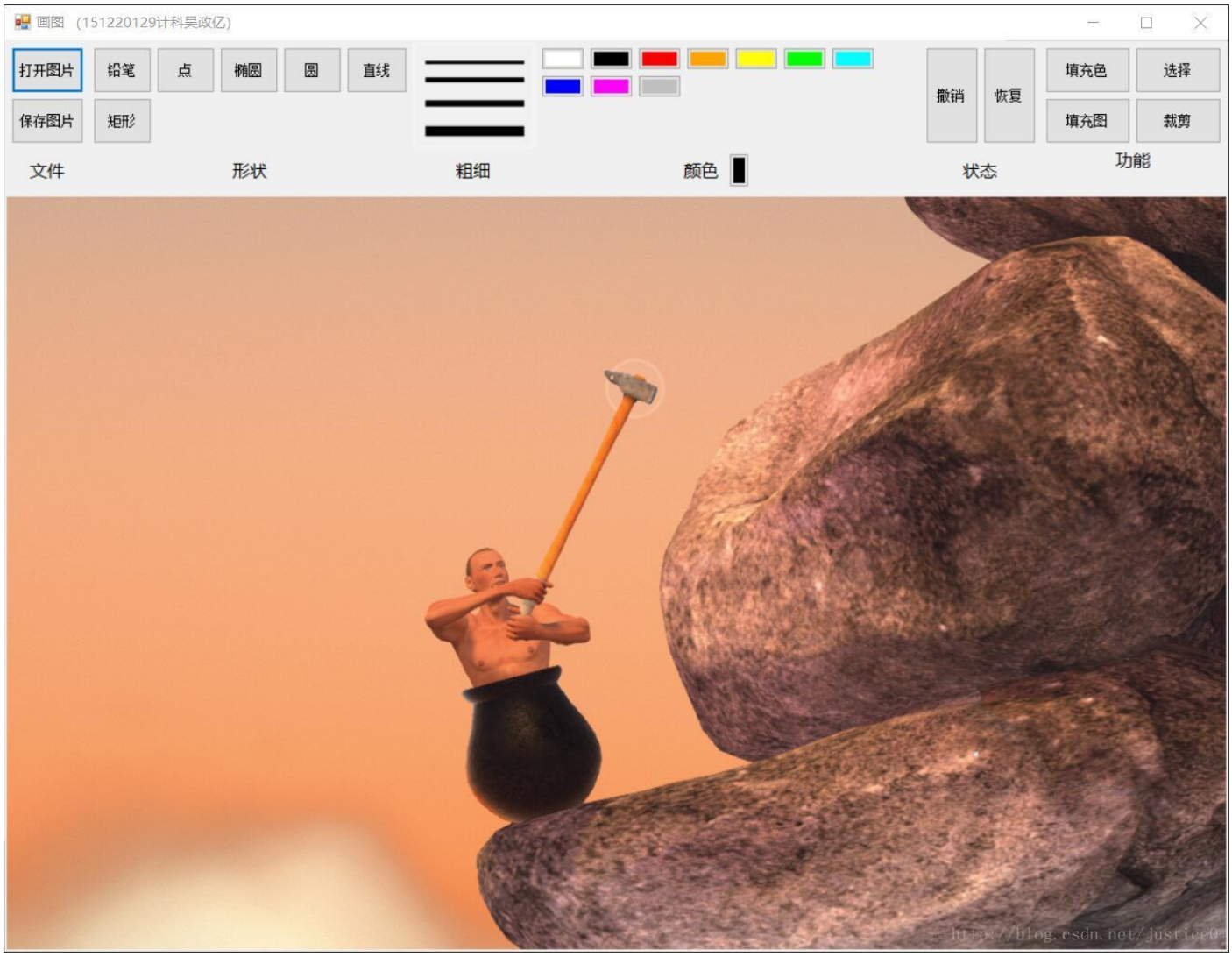
用C#向windows的画图看齐，实现了文件的打开与保存，直线/圆/曲线/点 等的形状绘制，可以自主选择粗细与颜色，并且应用派生的ArrayList类Step保存了每一步的操作，可以进行撤销与恢复操作。



十一月更新进度

实现了颜色填充，图片填充，选择，裁剪，平移等功能，并且引入了画布的概念 由于发现我的代码底层实现有问题，所以我在完成这次作业后，开始更改底层机理，因为之前我对每一步的操作存储的是操作执行后的图像，而这样的话想要单独对已经画过的图形进行重

新选择就变的十分困难。因此在新的重构的代码中我将保存的信息更改为每一次执行的操作(Shape)。



实验环境

报告采用markdown编写，并另存为了html与pdf格式，置于压缩包内report文件夹，code文件夹中包含了本次阶段性代码，Paint文件则是project的打包发布版本，便于助教测试。

基于Visual Studio 2015下的 C# 编写。

代码结构浅析

代码主要分为三个部分：

File Name	Information
`Program.cs`	应用程序的主入口点。
`Form1.cs`	画图程序的主窗口。

File Name	Information
Button_Click.cs	定义的颜色、功能按钮事件
Fill.cs	填充部分的实现
PictureBox.cs	画图的基本架构
Shape.cs	定义了各种形状的实现代码
Size.cs	对于调整画布、选框大小的实现
StepPaint.cs	ArrayList的派生类，用于存储每一步的操作。

Variable Name	Information
CASE NowCase	定义了当前的操作形状，例如画线与画圆
BREATH bh	定义了画笔的粗细
Color color	当前画笔颜色
int x0,y0	定义了直线，圆等的起点坐标，在鼠标摁下时更新
int x1,y1,x2,y2	定义了矩形选框的左上、右下角坐标
bool IsMouseDown	用于判断鼠标时候松开
bool IsBack	用于判断是否处于撤销状态
StepPaint Step	存储每一步的操作动作
Point p	用于调整Size时的临时变量
Bitmap ChoseRegion	存储被选择的区域图像
Button pictureBoxSize	用于调整pictureBox大小的按钮
Button ChoseSize	用于调整选区大小的按钮

Function Name	Information
private void InitForm1()	初始化成员变量
private void InitButton()	初始化按钮信息

Function Name	Information
`public Form1()`	Form1构造函数

Button_Click.cs

Function Name	Information
`private void button_*Color*_Click(...)`	不同颜色对应了不同的按钮来改变当前画笔颜色
`private void *Breath*ToolStripMenuItem_Click(...)`	设置画笔粗细的响应函数
`private void openfile_Click(...)`	打开文件响应函数
`private void savefile_Click(...)`	保存文件响应函数
`private void button_*Function*_Click(...)`	相应功能的相应函数，其中有撤销、恢复、填充、裁剪等
`private void button_*Shape*_Click(...)`	选择各个形状的相应函数
`private void CaseChange(CASE temp)`	更改当前状态

Fill.cs

Function Name	Information
`private void FillColor(...)`	在坐标(x,y)处用color泛滥填充
`private void FillPic(...)`	在(x0,y0),(x1,y1)界定的区域内用pic图像填充

PictureBox.cs

Function Name	Information
`private void pictureBox1_MouseUp(sender, e)`	鼠标左键松开时响应函数
`private void pictureBox1_MouseMove(sender, e)`	鼠标移动时响应函数
`private void pictureBox1_MouseDown(sender, e)`	鼠标左键摁下时响应函数

Shape.cs

Function Name	Information
`private void drawPixel(x,y,*color*)`	在(x,y)处画点(粗细取决于bh,颜色color可选)
`private void DDALine(x1,y1,x2,y2)`	画线函数DDA算法
`private void DDADottedLine(x1,y1,x2,y2)`	画 **虚线**函数DDA算法
`private void BresenhamLine(x1, y1, x2, y2)`	画线函数Bresenham算法
`private void MidpointLine(x0, y0, x1, y1)`	画线函数中心点生成算法
`private void BresenhamCircle(R, xc, yc)`	画圆函数Bresenham算法
`private void DrawRectangle(x0, y0, x1, y1)`	画矩形函数
`private void Bresenhamellipse(rx, ry, xc, yc)`	画椭圆函数Bresenham算法

Size.cs

Function Name	Information
`private void PictureBoxSize_Mouse*Up/Move/Down*(...)`	为了定义调整画布大小按钮的响应函数
`private void ChoseSize_Mouse*Up/Move/Down*(...)`	为了定义调整选区大小按钮的响应函数
`private void RefreshChoseSize(...)`	调整选区大小后刷新选区显示内容
`private void RefreshPicutreBoxSize(...)`	调整画布大小后刷新选区显示内容
`private bool PointInRectangle(int x, int y)`	判断点(x,y)是否在选区内
`private void PanningChoseRegion(int dx, int dy)`	平移选择的区域

StepPoint.cs

Function Name	Information
`ArrayList this`	存储所有步骤的Image
`int StepImage_now`	记录当前显示的图像在ArrayList的下标
`void ClearStep()`	清空ArrayList与StepImage_now
`void InitStep(Image)`	用一张Image初始化并作为保卫者
`void AddStep(obj)`	添加一个Image Step
`bool StepIsNull()`	判断StepImage_now是否位于最底部
`bool StepIsFull()`	判断StepImage_now是否位于最顶部
`Image PopStep()`	撤销当前步骤
`Image PushStep()`	恢复上一个步骤
`Image RefreshStep()`	将最后一个步骤永久撤销
`void RemoveNullStep()`	当撤销状态下执行一个步骤，将清除后面的步骤

核心代码实现

纯色泛滥填充算法[四连通]

一开始是用递归写的，发现会堆栈溢出，因此改为非递归算法，该算法用栈实现，每一次填充颜色成功便把自己周围的四个点入栈，

```
private void FillColor(Stack<point> pointStack, Bitmap map, Color OldColor, int x,
{
    point temp;
    while (pointStack.Count != 0)
    {
        temp = pointStack.Pop();
        //边界检测
        if (temp.x < 0 || temp.y < 0 || temp.x >= map.Width || temp.y >= map.Height
            continue;
        if (map.GetPixel(temp.x, temp.y) == OldColor && map.GetPixel(temp.x, temp.y
        {
            map.SetPixel(temp.x, temp.y, color);
            temp.x++;
            pointStack.Push(temp);
```

```
            temp.x -= 2;
            pointStack.Push(temp);
            temp.x++;
            temp.y++;
            pointStack.Push(temp);
            temp.y -= 2;
            pointStack.Push(temp);
        }
    }
}
```

图像填充算法

```
private void FillPic(Bitmap pic, int x0, int y0, int x1, int y1)
{
    Bitmap map = new Bitmap(pictureBox.Image);
    for (int i = x0; i < x1; i++)
    {
        for (int j = y0; j < y1; j++)
        {
            if (i - x0 > 0 && i - x0 < map.Width && j - y0 > 0 && j - y0 < map.Heig
                map.SetPixel(i, j, pic.GetPixel(i - x0, j - y0));
        }
    }
    pictureBox.Image = map;
}
```

图像拉伸算法

先求得缩放系数，然后通过计算获得拉伸后的坐标与颜色信息

```
static public Bitmap SetBitmapSize(Bitmap src, int w, int h)
{
    int tw = src.Width;
    int th = src.Height;
    double dw = (double)w / tw;
    double dh = (double)h / th;
    Bitmap map = new Bitmap(w, h);
    for (int i = 0; i < w; i++)
    {
        for (int j = 0; j < h; j++)
        {
            map.SetPixel(i, j, src.GetPixel((int)((double)i / dw), (int)((double)j
```

```
            }
        }
        return map;
    }
```

实验感悟

这一次实验，我添加了矩形与椭圆，并且实现了平移，缩放，填充，裁剪等功能。 在实现椭圆代码时，发现PPT给的信息有误，导致DeBug许久有点难受，最后自己算了一遍所有参数。 本次实验实现后，觉的我存储步骤的方法信息不够全面(之前是存下了每一次步骤后的Image)， 我现在已经着手开始重构我的代码，重构部分目前已经实现了各种形状的绘画，其他部分仍在实现中。