電腦圖學 期中考作業(110 上)

> 408261292 資工三甲 丁柏瑋

作業題目

　修改所附期中考替代作業參考資料(110 上)資料夾內作業參考程式: newpaint2.c 這個陽春版小畫家程式, 加入下列 幾個功能:

- [x] ~~畫多邊形(增加)/刪除多邊形~~
- [x] ~~噴槍~~
- [x] ~~橡皮擦~~
- [ ] 以橡皮筋方式來畫長方形、圓形
- [x] ~~原有之顯示英文字串功能新增可以有幾種字型選擇~~

　做了以上所有功能, 就有基本成績 75 分, 鼓勵同學自行增加其它功能(可參考 Windows 內小畫家現有功能)或發揮創意 新增功能(如:畫曲線, 畫隱藏面去除之 3D 模型,存檔,再載入之前檔案 繼續編輯等等其他功能), 讓你所寫的小畫家功能愈多愈完整, 此項期中考替代作業成績會適度的加分, 滿分 100 分。

新增功能

- 顏色統一選擇器

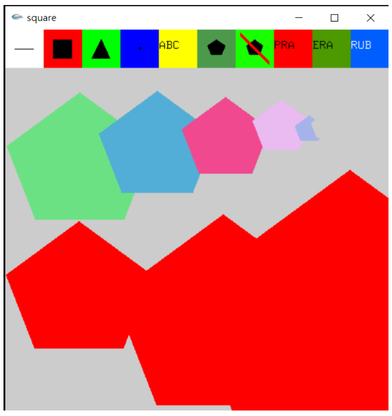
  可以選擇標準顏色進行統一輸出。
- 橡皮擦大小選擇器

  控制橡皮擦大小，或是重設
- 多邊形大小選擇器

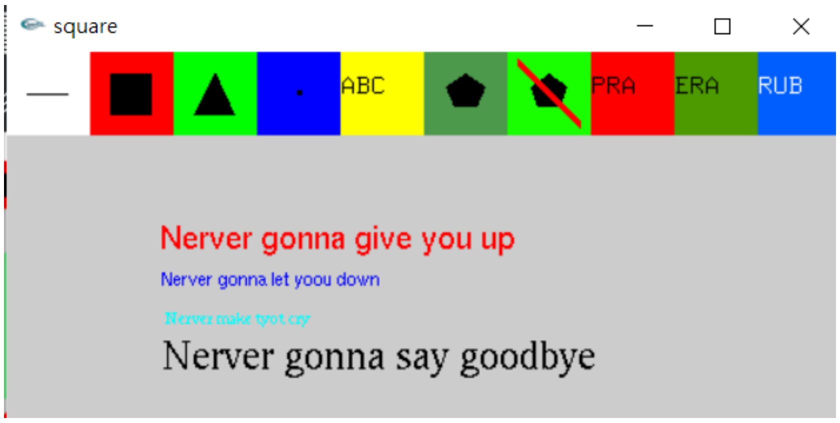
  控制多邊形大小，或是重設。且刪除時自動記錄大小。

預覽畫面

1. 多邊形大小選擇與顏色選擇器



2. 利用文字功能讓每段文字都有特色



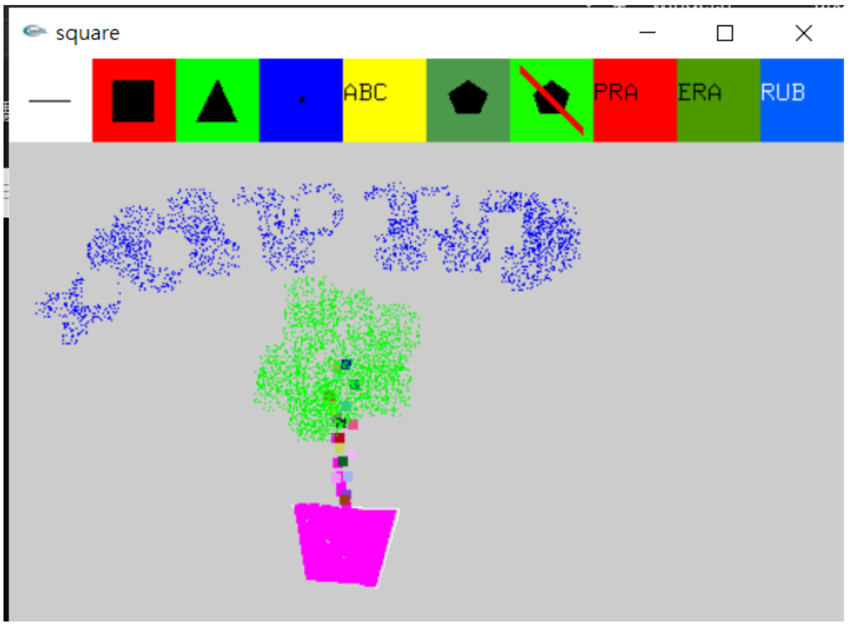
Nerver gonna give you up

Nerver gonna let yoou down

Nerver make tyot cry

Nerver gonna say goodbye

3. 功能實現3



程式架構

將相似功能盡量使用同一個函數，並把共用數據放到public使數據統一存取。

系統類

void mouse(int, int, int, int);
void key(unsigned char, int, int);
void display(void);

void myinit(void);
void myReshape(GLsizei, GLsizei);

交互類

void screen_box(int, int, int);
void right_menu(int);
void middle_menu(int);
void color_menu(int);
void font_menu(int);

void pixel_menu(int);
void fill_menu(int);

int pick(int, int);

功能類

void getColor(int);
void getFont(unsigned char);

顯示類

void drawSquare(int, int);
void drawPentagon(int, int);
void remove_Pentagon();
void sprayGun(int, int);
void eraser(int, int);
void rubber(int, int);

討論

很複雜，花很多時間。不過不是很難，橡皮筋功能想不到實現方法。

程式碼

```
/* This program illustrates the use of the glut library for
interfacing with a window system */


#define NULL 0
#define LINE 1
#define RECTANGLE 2
#define TRIANGLE  3
#define POINTS 4
#define TEXT 5
#define PENTAGON 6
#define ERASE_PENTAGON 7
#define PRASE 8
#define ERASER 9
#define RUBBER 10

#define MAXPENTA 256

#include <stdio.h>
#include <GL/glut.h>



void mouse(int, int, int, int);
void key(unsigned char, int, int);
void display(void);

//tool
void drawSquare(int, int);
void drawPentagon(int, int);
void remove_Pentagon();
void sprayGun(int, int);
void eraser(int, int);
void rubber(int, int);

void myinit(void);
void myReshape(GLsizei, GLsizei);
void screen_box(int, int, int);
void right_menu(int);
void middle_menu(int);
void color_menu(int);
void font_menu(int);
void pixel_menu(int);
void fill_menu(int);
```

```
int pick(int, int);
void getColor(int);
void getFont(unsigned char);

/* globals */

GLsizei wh = 500, ww = 500; /* initial window size */
GLfloat size = 3.0;   /* half side length of square */
int draw_mode = 0; /* drawing mode */
int rx, ry; /*raster position*/
GLfloat r = 1.0, g = 1.0, b = 1.0; /* drawing color */



int fill = 0; /* fill flag */
int fontType = 1;
char* rubberType;
int pentagonCount = 0;
int nowColor = 0;
float eraserSize = 10;

//penta const
float h = 30;
float c = 66;
float d = 58;
float a = 95.3;
float pentaSize = 100;

//penta struct
struct PentaCollect
{
    int x, y;
    float psSize;
} pentaCollect[MAXPENTA];
//tool function

void getColor(int id)
{
    /*
    0   : Ramdom(Default)
    if(id == 1) {r = 1.0; g = 0.0; b = 0.0;}
    else if(id == 2) {r = 0.0; g = 1.0; b = 0.0;}
    else if(id == 3) {r = 0.0; g = 0.0; b = 1.0;}
    else if(id == 4) {r = 0.0; g = 1.0; b = 1.0;}
    else if(id == 5) {r = 1.0; g = 0.0; b = 1.0;}
    else if(id == 6) {r = 1.0; g = 1.0; b = 0.0;}
    else if(id == 7) {r = 1.0; g = 1.0; b = 1.0;}
    else if(id == 8) {r = 0.0; g = 0.0; b = 0.0;}

    99   :background
    */
    switch (id)
    {
    case 0:
        glColor3ub((char)rand() % 256, (char)rand() % 256, (char)rand() % 256);
        break;
    case 1:
        glColor3f(1.0f, 0.0f, 0.0f);
        break;
    case 2:
        glColor3f(0.0f, 1.0f, 0.0f);
        break;
    case 3:
        glColor3f(0.0f, 0.0f, 1.0f);
        break;
    case 4:
        glColor3f(0.0f, 1.0f, 1.0f);
        break;
    case 5:
        glColor3f(1.0f, 0.0f, 1.0f);
        break;
    case 6:
        glColor3f(1.0f, 1.0f, 0.0f);
        break;
    case 7:
        glColor3f(1.0f, 1.0f, 1.0f);
        break;
    case 8:
        glColor3f(0.0f, 0.0f, 0.0f);
        break;
    case 99:
        glColor3f(0.8f, 0.8f, 0.8f);
        break;
    default:
        glColor3ub((char)rand() % 256, (char)rand() % 256, (char)rand() % 256);
    }
}
```

```
void drawSquare(int x, int y)
{
    y = wh - y;
    getColor(nowColor);
    glBegin(GL_POLYGON);
    glVertex2f(x + size, y + size);
    glVertex2f(x - size, y + size);
    glVertex2f(x - size, y - size);
    glVertex2f(x + size, y - size);
    glEnd();
}

void drawPentagon(int x, int y)
{
    if (wh - y + pentaSize > wh - ww / 10)
    {
        return;
    }
    y = wh - y;
    float sca = pentaSize / 100;
    float ta = a * sca;
    float th = h * sca;
    float tc = c * sca;
    float td = d * sca;
    getColor(nowColor);
    glBegin(GL_POLYGON);
    glVertex2f(x, y + pentaSize);
    glVertex2f(x + ta, y + th);
    glVertex2f(x + td, y - tc);
    glVertex2f(x - td, y - tc);
    glVertex2f(x - ta, y + th);
    glEnd();
    pentaCollect[pentagonCount].x = x;
    pentaCollect[pentagonCount].y = y;
    pentaCollect[pentagonCount].psSize = pentaSize;
    pentagonCount++;
}


void remove_Pentagon()
{
    if (pentagonCount == 0)
        return;
    pentagonCount--;
    float tmpSize = pentaSize;
    int tmpNowcolor = nowColor;

    pentaSize = pentaCollect[pentagonCount].psSize;
    nowColor = 99;

    //copy form drawPentagon
    float tx = pentaCollect[pentagonCount].x;
    float ty = pentaCollect[pentagonCount].y;
    float sca = pentaSize / 100;
    float ta = a * sca;
    float th = h * sca;
    float tc = c * sca;
    float td = d * sca;
    getColor(nowColor);
    glBegin(GL_POLYGON);
    glVertex2f(tx, ty + pentaSize);
    glVertex2f(tx + ta, ty + th);
    glVertex2f(tx + td, ty - tc);
    glVertex2f(tx - td, ty - tc);
    glVertex2f(tx - ta, ty + th);
    glEnd();
    //copy end

    pentaSize = tmpSize;
    nowColor = tmpNowcolor;
}

/* geronn sareta ketenn */
void sprayGun(int x, int y)
{
    getColor(nowColor);
    int outcount = 0, tmp;
    for (int i = x - 15; i <= x + 15; i++)
    {
        for (int j = y - 15; j <= y + 15; j++)
        {
            if (outcount < 300 && (i >= x - 10 || j >= y - 10) && (i >= x - 10 || j <= y + 10) && (i <= x + 10 || j >= y - 10) && (i <
            {
                tmp = rand() % 5;  //dot Den
                if (tmp == 0)
                {
                    glBegin(GL_POINTS);
                    glVertex2i(i, wh - j);
                    glEnd();

                    outcount++;
                }
            }
        }
    }
}


void eraser(int x, int y)
{
    y = wh - y;
    getColor(99);   //BG color
    glBegin(GL_POLYGON);
    glVertex2f(x + eraserSize, y + eraserSize);
    glVertex2f(x - eraserSize, y + eraserSize);
    glVertex2f(x - eraserSize, y - eraserSize);
    glVertex2f(x + eraserSize, y - eraserSize);
    glEnd();
}

void rubber(int x, int y)
{
    //todo
}


/* rehaping routine called whenever window is resized
or moved */

void myReshape(GLsizei w, GLsizei h)
{

    /* adjust clipping box */

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, (GLdouble)w, 0.0, (GLdouble)h, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    /* adjust viewport and  clear */

    glViewport(0, 0, w, h);
    glClearColor(0.8, 0.8, 0.8, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    display();
    glFlush();

    /* set global size for use by drawing routine */

    ww = w;
    wh = h;
}

void myinit(void)
{

    glViewport(0, 0, ww, wh);


    /* Pick 2D clipping window to match size of X window
    This choice avoids having to scale object coordinates
    each time window is resized */

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, (GLdouble)ww, 0.0, (GLdouble)wh, -1.0, 1.0);

    /* set clear color to black and clear window */

    glClearColor(0.8, 0.8, 0.8, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}



void mouse(int btn, int state, int x, int y)
{
    static int count;
    int where;
    static int xp[2], yp[2];
    if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        glPushAttrib(GL_ALL_ATTRIB_BITS);

        where = pick(x, y);
        glColor3f(r, g, b);

        //ERASE_PENTAGON mode setting

        if (where == ERASE_PENTAGON)
        {
            remove_Pentagon();
            //reset
            count = 0;
            draw_mode = 0;
        }

        if (where != 0)
        {

            count = 0;
            draw_mode = where;
        }
        else switch (draw_mode)
        {
        case(LINE):
            if (count == 0)
            {
                count++;
                //save point 1
                xp[0] = x;
                yp[0] = y;
            }
            else
            {
                getColor(nowColor);
                glBegin(GL_LINES);
                glVertex2i(x, wh - y);
                glVertex2i(xp[0], wh - yp[0]);
                glEnd();
                draw_mode = 0;
                count = 0;
            }
            break;
        case(RECTANGLE):
            if (count == 0)
            {
                count++;
                xp[0] = x;
                yp[0] = y;
            }
            else
            {
                getColor(nowColor);
                if (fill) glBegin(GL_POLYGON);
                else glBegin(GL_LINE_LOOP);
                glVertex2i(x, wh - y);
                glVertex2i(x, wh - yp[0]);
                glVertex2i(xp[0], wh - yp[0]);
                glVertex2i(xp[0], wh - y);
                glEnd();
                draw_mode = 0;
                count = 0;
            }
            break;
        case (TRIANGLE):
            switch (count)
            {
            case(0):
                count++;
                xp[0] = x;
                yp[0] = y;
                break;
            case(1):
                count++;
                xp[1] = x;
                yp[1] = y;
                break;
            case(2):
                getColor(nowColor);
                if (fill) glBegin(GL_POLYGON);
                else glBegin(GL_LINE_LOOP);
                glVertex2i(xp[0], wh - yp[0]);
                glVertex2i(xp[1], wh - yp[1]);
                glVertex2i(x, wh - y);
                glEnd();
                //reset
                draw_mode = 0;
                count = 0;
            }
            break;
        case(POINTS):
        {
            drawSquare(x, y);
            count++;
        }
        break;
        case(TEXT):
        {
            rx = x;
            ry = wh - y;
            glRasterPos2i(rx, ry);
            count = 0;
        }
        break;
        case(PENTAGON):
            drawPentagon(x, y);
            break;
        case(PRASE):
            sprayGun(x, y);
            break;
        case(ERASER):
            eraser(x, y);
            break;
        case(RUBBER):
            break;

        default:
            break;
        }

        glPopAttrib();
        glFlush();
    }
}

int pick(int x, int y)
{
    y = wh - y;
    if (y < wh - ww / 10) return 0;
    else if (x < 1 * ww / 10) return LINE;
    else if (x < 2 * ww / 10) return RECTANGLE;
    else if (x < 3 * ww / 10) return TRIANGLE;
    else if (x < 4 * ww / 10) return POINTS;
    else if (x < 5 * ww / 10) return TEXT;
    else if (x < 6 * ww / 10) return PENTAGON;
    else if (x < 7 * ww / 10) return ERASE_PENTAGON;
    else if (x < 8 * ww / 10) return PRASE;
    else if (x < 9 * ww / 10) return ERASER;
    else return 0;
}


void screen_box(int x, int y, int s)
{
    glBegin(GL_QUADS);
    glVertex2i(x, y);
    glVertex2i(x + s, y);
    glVertex2i(x + s, y + s);
    glVertex2i(x, y + s);
    glEnd();
}

void right_menu(int id)
{
    if (id == 1) exit(0);
    else
    {
        display();
        pentagonCount = 0;   //defence overclean
    }
}

void middle_menu(int id)
{
    //exective
}

void color_menu(int id)
{
    nowColor = id;
}
```

```
void pixel_menu(int id)
{
    if (id == 1) size = 2 * size;
    else if (size > 1) size = size / 2;
}

void font_menu(int id)
{
    fontType = id;
}

void rubber_menu(int id)
{
    switch (id)
    {
    case 1:
        rubberType = "rectangle";
        break;
    case 2:
        rubberType = "circle";
        break;
    default:
        printf("Explicit at rubber_menu");
        break;
    }
}

void fill_menu(int id)
{
    if (id == 1) fill = 1;
    else fill = 0;
}

void pentaSize_menu(int id)
{
    if (id == 0)
    {
        pentaSize -= 20;
        if (pentaSize < 20)
        {
            pentaSize = 20;
        }
    }
    else if (id == 1)
    {
        pentaSize *= 1.5;
    }
    else
    {
        pentaSize = 100;    //reset
    }
}

eraserSize_menu(int id)
{
    if (id == 0)
    {
        if (eraserSize < 3)
        {
            eraserSize = 3;
        }
        eraserSize *= 0.8;
    }
    else if (id == 1)
    {
        eraserSize *= 1.2;
    }
    else
    {
        eraserSize = 10;    //reset
    }
}

void key(unsigned char c, int xx, int yy)
{
    if (draw_mode != TEXT) return;
    getColor(nowColor);
    glRasterPos2i(rx, ry);
    getFont(c);

}

void getFont(unsigned char c)
{
    switch (fontType)
    {
    case 1:
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_10, c);
        rx += glutBitmapWidth(GLUT_BITMAP_HELVETICA_10, c);
        break;
    case 2:
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, c);
        rx += glutBitmapWidth(GLUT_BITMAP_HELVETICA_18, c);
        break;
    case 3:
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_10, c);
        rx += glutBitmapWidth(GLUT_BITMAP_TIMES_ROMAN_10, c);
        break;
    case 4:
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, c);
        rx += glutBitmapWidth(GLUT_BITMAP_TIMES_ROMAN_24, c);
        break;
    default:
        break;
    }
}

//display the tools bar
void display(void)
{
    int shift = 0;
    glPushAttrib(GL_ALL_ATTRIB_BITS);
    glClearColor(0.8, 0.8, 0.8, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    screen_box(0, wh - ww / 10, ww / 10);
    glColor3f(1.0, 0.0, 0.0);
    screen_box(ww / 10, wh - ww / 10, ww / 10);
    glColor3f(0.0, 1.0, 0.0);
    screen_box(2 * ww / 10, wh - ww / 10, ww / 10);
    glColor3f(0.0, 0.0, 1.0);
    screen_box(3 * ww / 10, wh - ww / 10, ww / 10);
    glColor3f(1.0, 1.0, 0.0);
    screen_box(4 * ww / 10, wh - ww / 10, ww / 10);
    glColor3f(0.3, 0.6, 0.3);
    screen_box(5 * ww / 10, wh - ww / 10, ww / 10);
    glColor3f(0.1, 1.0, 0.0);
    screen_box(6 * ww / 10, wh - ww / 10, ww / 10);
    glColor3f(1.0, 0.0, 0.0);
    screen_box(7 * ww / 10, wh - ww / 10, ww / 10);
    glColor3f(0.3, 0.6, 0.0);
    screen_box(8 * ww / 10, wh - ww / 10, ww / 10);
    glColor3f(0.0, 0.37, 1.0);
    screen_box(9 * ww / 10, wh - ww / 10, ww / 10);
    //under box
    glColor3f(0.0, 0.0, 0.0);
    screen_box(ww / 10 + ww / 40, wh - ww / 10 + ww / 40, ww / 20);
    //Line
    glBegin(GL_LINES);
    glVertex2i(wh / 40, wh - ww / 20);
    glVertex2i(wh / 40 + ww / 20, wh - ww / 20);
    glEnd();
    //Traingle
    glBegin(GL_TRIANGLES);
    glVertex2i(ww / 5 + ww / 40, wh - ww / 10 + ww / 40);
    glVertex2i(ww / 5 + ww / 20, wh - ww / 40);
    glVertex2i(ww / 5 + 3 * ww / 40, wh - ww / 10 + ww / 40);
    glEnd();
    glPointSize(3.0);
    glBegin(GL_POINTS);
    glVertex2i(3 * ww / 10 + ww / 20, wh - ww / 20);
    glEnd();
    glRasterPos2i(2 * ww / 5, wh - ww / 20);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'A');
    shift = glutBitmapWidth(GLUT_BITMAP_9_BY_15, 'A');
    glRasterPos2i(2 * ww / 5 + shift, wh - ww / 20);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'B');
    shift += glutBitmapWidth(GLUT_BITMAP_9_BY_15, 'B');
    glRasterPos2i(2 * ww / 5 + shift, wh - ww / 20);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'C');
    // draw polygon

    glBegin(GL_POLYGON);
    glVertex2i((11 * ww / 20), (wh - ww / 20) + ww / 40);
    glVertex2i((11 * ww / 20) + ww / 40, (wh - ww / 20) + ww / 125);
    glVertex2i((11 * ww / 20) + ww / 70, (wh - ww / 20) - ww / 60);
    glVertex2i((11 * ww / 20) - ww / 70, (wh - ww / 20) - ww / 60);
    glVertex2i((11 * ww / 20) - ww / 40, (wh - ww / 20) + ww / 125);
    glEnd();
    //erase polygon
    //unit => ww/40[1|2]
    glBegin(GL_POLYGON);
    glVertex2i((13 * ww / 20), (wh - ww / 20) + ww / 40);
    glVertex2i((13 * ww / 20) + ww / 42, (wh - ww / 20) + ww / 125);
    glVertex2i((13 * ww / 20) + ww / 71, (wh - ww / 20) - ww / 60);
    glVertex2i((13 * ww / 20) - ww / 71, (wh - ww / 20) - ww / 60);
    glVertex2i((13 * ww / 20) - ww / 42, (wh - ww / 20) + ww / 125);
    glEnd();

    glLineWidth(5.0);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_LINES);
    glVertex2i(6 * ww / 10 + ww / 80, wh - ww / 80);
    glVertex2i(7 * ww / 10 - ww / 80, wh - ww / 10 + ww / 80);
    glEnd();
    //prase
    shift = 0;
    getColor(8);    //black
    glRasterPos2i(7 * ww / 10, wh - ww / 20);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'P');
    shift += glutBitmapWidth(GLUT_BITMAP_9_BY_15, 'P');
    glRasterPos2i(7 * ww / 10 + shift, wh - ww / 20);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'R');
    shift += glutBitmapWidth(GLUT_BITMAP_9_BY_15, 'R');
    glRasterPos2i(7 * ww / 10 + shift, wh - ww / 20);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'A');

    //Eraser
    shift = 0;
    getColor(8);    //black
    glRasterPos2i(8 * ww / 10, wh - ww / 20);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'E');
    shift += glutBitmapWidth(GLUT_BITMAP_9_BY_15, 'E');
    glRasterPos2i(8 * ww / 10 + shift, wh - ww / 20);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'R');
    shift += glutBitmapWidth(GLUT_BITMAP_9_BY_15, 'R');
    glRasterPos2i(8 * ww / 10 + shift, wh - ww / 20);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'A');

    //Rubber
    shift = 0;

    glColor3f(1.0, 1.0, 1.0);           //todo
    glRasterPos2i(9 * ww / 10, wh - ww / 20);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'R');
    shift += glutBitmapWidth(GLUT_BITMAP_9_BY_15, 'R');
    glRasterPos2i(9 * ww / 10 + shift, wh - ww / 20);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'U');
    shift += glutBitmapWidth(GLUT_BITMAP_9_BY_15, 'U');
    glRasterPos2i(9 * ww / 10 + shift, wh - ww / 20);
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15, 'B');


    glFlush();
    glPopAttrib();
}


int main(int argc, char** argv)
{
    int c_menu, p_menu, f_menu, ft_menu, r_menu, ps_menu, e_menu;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("square");

    glutDisplayFunc(display);

    //set about menu select
    c_menu = glutCreateMenu(color_menu);
    glutAddMenuEntry("Ramdom", 0);
    glutAddMenuEntry("Red", 1);
    glutAddMenuEntry("Green", 2);
    glutAddMenuEntry("Blue", 3);
    glutAddMenuEntry("Cyan", 4);
    glutAddMenuEntry("Magenta", 5);
    glutAddMenuEntry("Yellow", 6);
    glutAddMenuEntry("White", 7);
    glutAddMenuEntry("Black", 8);
    p_menu = glutCreateMenu(pixel_menu);
    glutAddMenuEntry("increase pixel size", 1);
    glutAddMenuEntry("decrease pixel size", 2);
    f_menu = glutCreateMenu(fill_menu);
    glutAddMenuEntry("fill on", 1);
    glutAddMenuEntry("fill off", 2);
    ps_menu = glutCreateMenu(pentaSize_menu);
    glutAddMenuEntry("small", 0);
    glutAddMenuEntry("big", 1);
    glutAddMenuEntry("reset", 2);
    e_menu = glutCreateMenu(eraserSize_menu);
    glutAddMenuEntry("small", 0);
    glutAddMenuEntry("big", 1);
    glutAddMenuEntry("reset", 2);

    ft_menu = glutCreateMenu(font_menu);
    glutAddMenuEntry("HELVETICA_10", 1);
    glutAddMenuEntry("HELVETICA_18", 2);
    glutAddMenuEntry("TIMES_ROMAN_10", 3);
    glutAddMenuEntry("TIMES_ROMAN_24", 4);

    r_menu = glutCreateMenu(rubber_menu);
    glutAddMenuEntry("Rectangle", 1);
    glutAddMenuEntry("Circle", 2);
    glutCreateMenu(right_menu);
    glutAddMenuEntry("quit", 1);
    glutAddMenuEntry("clear", 2);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutCreateMenu(middle_menu);
    glutAddSubMenu("Colors", c_menu);
    glutAddSubMenu("Pixel Size", p_menu);
    glutAddSubMenu("Pentagon Size", ps_menu);
    glutAddSubMenu("Eraser Size", e_menu);
    glutAddSubMenu("Fill", f_menu);
    glutAddSubMenu("Font", ft_menu);
    glutAddSubMenu("Rubber select", r_menu);
    glutAttachMenu(GLUT_MIDDLE_BUTTON);

    myinit();
    glutReshapeFunc(myReshape);
    glutKeyboardFunc(key);
    glutMouseFunc(mouse);
    glutMainLoop();

}
```