CHAPTER 1

INTRODUCTION

This is a program written in the OpenGL programming interface. The program shows an array of squares connected to each other. At first the all the blocks are shown. Various functions are used in the code to implement the game.

The array is altered according to the rules of the game depending on the square selected by the player. The motive of the game is to select the longest chain of adjacent squares of same color.

A track of the score is maintained based the number of squares eliminated while playing. When there are no more adjacent squares of same color the game ends and the final score is displayed.

CHAPTER 2

SYSTEM SPECIFICATION

2.1 Software Requirements

1.Operating System : Windows XP

2.Compiler Used : Visual C++ 6.0

3.Language Used : C language

4.Editor : vi editor

2.2 Hardware Requirements

1.Main processor : Pentium III

2.Processor Speed : 800 MHz

3.RAM Size : 128 MB DDR

4.Keyboard : Standard qwerty serial or PS/2 keyboard

5.Mouse : Standard serial or PS/2 mouse

6.Compatibility : AT/T Compatible

7.Cache memory : 256 KB

8.Diskette drive A : 1.44 MB, 3.5 inches

CHAPTER 3

Overview

3.1 Computer Graphics.

Computer Graphics is concerned with all aspects of producing pictures or images. The term computer graphics includes almost everything on computers that is not text or sound. Today nearly all computers use some graphics and users expect to control their computer through icons and pictures rather than just by typing. The term Computer Graphics has several meanings:

- the representation and manipulation of pictorial data by a computer
- the various technologies used to create and manipulate such pictorial data
- the images so produced, and
- the sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content, see study of computer graphics

Today computers and computer-generated images touch many aspects of our daily life. Computer imagery is found on television, in newspapers, in weather reports, and during surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. Such graphs are used to illustrate papers, reports, theses, and other presentation material. A range of tools and facilities are available to enable users to visualize their data, and computer graphics are used in many disciplines.

3.2 OpenGL Overview

OpenGL (***Open Graphics Library***) is a standard specification for writing applications that produce 2D and 3D computer graphics. That means you provide the data in an OpenGL-useable form and OpenGL will show this data on the screen (render it).

It is developed by many companies and it is free to use. You can develop OpenGL-applications without licensing. OpenGL is a hardware and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed implementation.

Because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

GLUT is a complete API written by Mark Kilgard which lets you create windows and handle the messages. It exists for several platforms, that means that a program which uses GLUT can be compiled on many platforms without (or at least with very few) changes in the code.
OpenGL is a state machine. You put it into various states (or modes) that then remain in effect until you change them. For example, the current color is a state variable. You can set

the current color to white, red or any color and thereafter every object is drawn with that color until you set the current color to something else. The current color is only one of many state variables that OpenGL maintains. Other control such things as the current viewing and projection transformation line and polygon stipple patterns, polygon drawing models, pixel-packing conventions, positions and characteristics of lights, and material properties of the objects being drawn.

3.3 OpenGL Rendering Pipeline

Most implementations of OpenGL have a similar order of operations, a series of

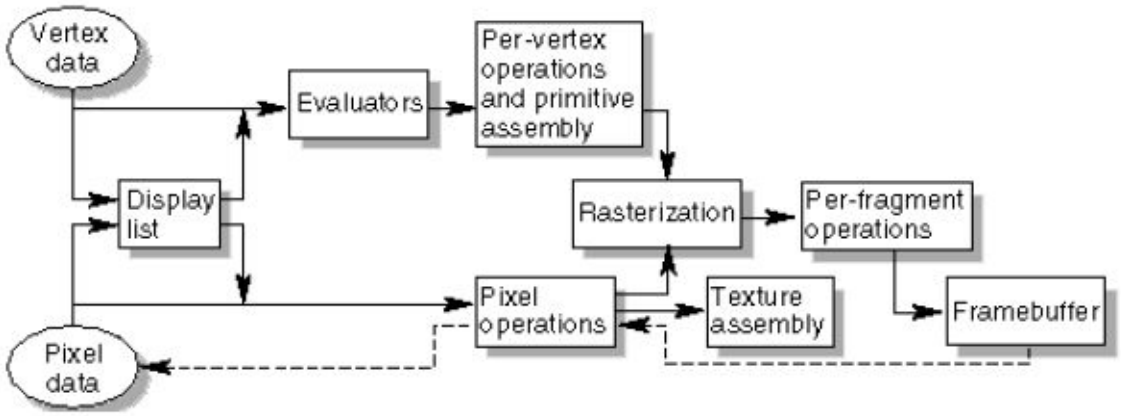


Figure 4.1 Order of Operations

processing stages called the OpenGL rendering pipeline.

Geometric data (vertices, lines, and polygons) follow the path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images, and bitmaps) are treated differently for part of the process. Both types of data undergo the same final steps (rasterization and per-fragment operations) before the final pixel data is written into the frame buffer.

CHAPTER 4

ANALYSIS and DESIGN

ANALYSIS:

The simulation involves analyzing the matrix of squares to find all the adjacent squares to the selected square. Depending on his choice those squares get eliminated and rest of the squares cascades towards the top left corner of the window.
We have provided a mouse interface for the selection of various options present in the game. The matrix of squares can also be altered by using the options available in the menu box.

DESIGN:

Design specification of Bubble Breaker

- A random matrix of squares is generated.
- Design the corresponding menu generated on right click of the mouse.
- Design the functions required to do the needed tasks.
- Determine the processing necessary to perform the task.

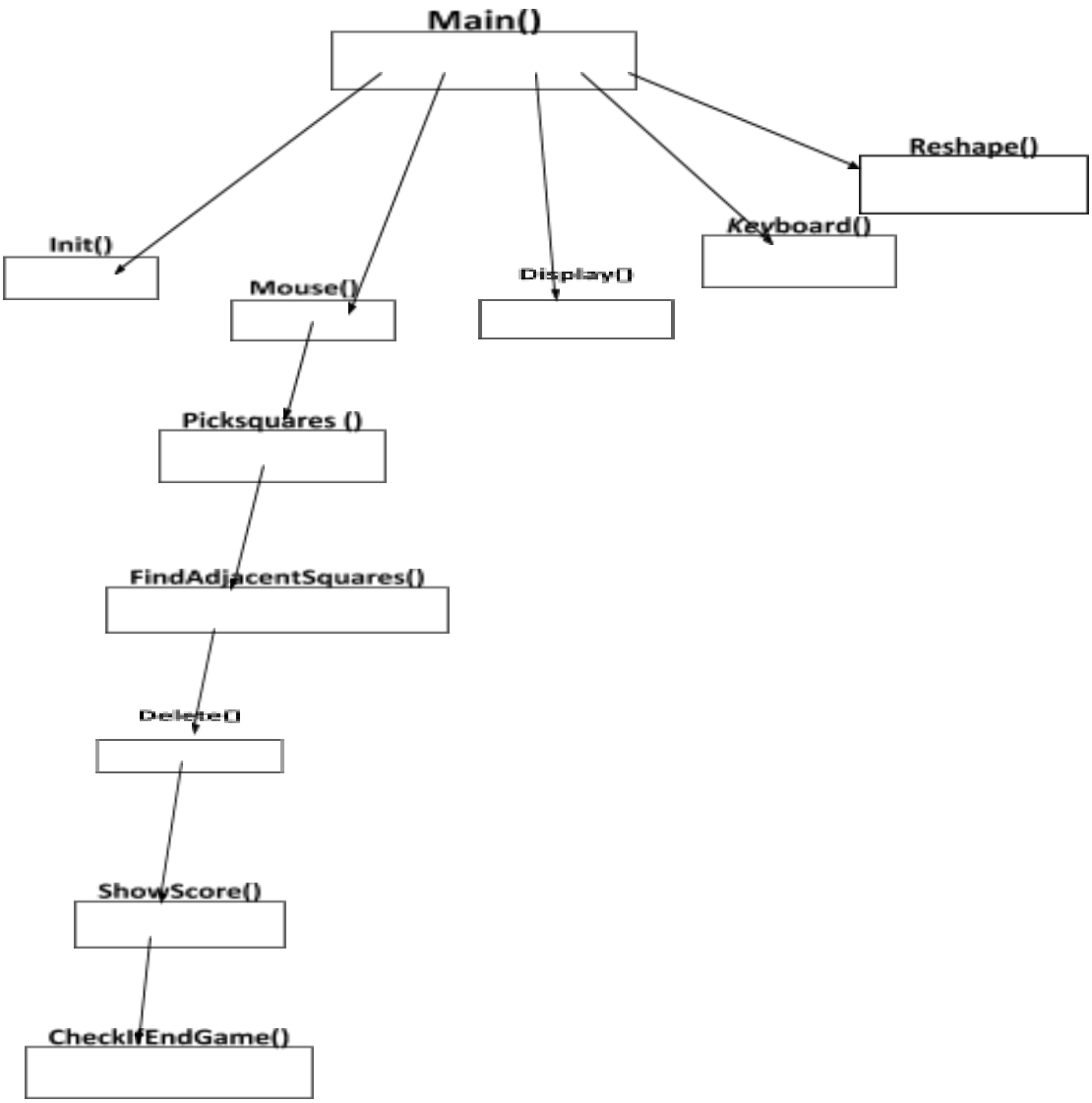
Design of Bubble Breaker

The design of square breaker involves generating a random array of squares which is done by init(). The corresponding rules of the game are implemented in a nested call of procedures initiated by the mouse function. We have also included a keyboard() and a reshape() that are used for exit and reshape the window respectively.

The functions used in my project are:

- Main function
- Init
- Display
- Keyboard

FLOWCHART


Main()
Reshape()
Init()
Keyboard()
Mouse()
Display()
Picksquares ()
FindAdjacentSquares()
Delete()
ShowScore()
CheckIfEndGame()

CHAPTER 5

IMPLEMENTATION

The OpenGL provides very powerful translation, rotation and scaling facilities which relive the programmers by allowing them to concentrate on their job rather than focusing on how to implement these operations. OpenGL also provides viewing and modeling transformations. Given below is our implementation code for Square Breaker:

```
/*
 * Square Breaker.cpp
 * AUTHORS Sunil Saurabh
 * This code simulates a game in which one has to look for ajacent squares and break them &
 * longer the chain the more you score.
 */
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
#include<math.h>
#include<time.h>
#include<stdlib.h>
#include<conio.h>
#include <stdarg.h>                                    // Header File For Variable Argument Routines  ( ADD )
#include<string.h>
#include <windows.h>

 int board[6][6]={1,1,1,1,2,1,       //this matrix hold the current value of the colors
                  0,1,0,0,2,1,
                                    0,1,1,0,2,1,
                                    0,1,0,0,2,1,
                                    0,1,0,0,2,1,
                                     0,1,1,0,2,1,
                                    };
int  mark[6][6]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};// It holds the position that are adjacent to the square we have selected
int count=0;
int tempbrd[6][6],tempmark[6][6];
int totalscore=0;
int end=0;
void Sprint( int x, int y, char *st)
{
        int l,i;

        l=strlen( st ); // see how many characters are in text string.
```

```
        glRasterPos2i( x, y); // location to start printing text
        for( i=0; i < l; i++)  // loop until i is greater then l
                {
                glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, st[i]); // Print a
character on the screen
        }

}

/*  Clear color value for every square on the board   */

void init(void)
{
                srand((unsigned)time(NULL));
  int i, j;

    for (i = 0; i < 6; i++)
   for (j = 0; j < 6; j ++)
       board[i][j] = (rand())%3;
  glClearColor (1.0, 1.0, 1.0, 1.0);
}

/*  The nine squares are drawn.  In selection mode, each
 *  square is given two names:  one for the row and the
 *  other for the column on the grid.  The color of each
 *  square is determined by its position on the grid, and
 *  the value in the board[][] array.
 */
void drawSquares(GLenum mode)
{
  GLuint i, j;
   for (i = 0; i <6; i++) {

     if (mode == GL_SELECT)
       glLoadName (i);
     for (j = 0; j <6; j ++) {
       if (mode == GL_SELECT)
         glPushName (j);
              if(board[i][j]==1)
           {
       glColor3f (1.0,0.0,0.0);
                  glRectf (j+.02, i+.02, j+.98, i+.98);
           }
            if(board[i][j]==2)
           {
                    glColor3f (0.0,1.0,0.0);
                  glRectf (j+.02, i+.02, j+.98, i+.98);
           }
```

```
            if(board[i][j]==0)
            {
                    glColor3f(0.0,0.0,1.0);
                   glRectf (j+.02, i+.02, j+.98, i+.98);
            }
         if (mode == GL_SELECT)
           glPopName ();
     }
   }
   for(i=0;i<6;i++)
            for(j=0;j<6;j++)
            {
                    int temp;
                    temp=board[i][j];
                    board[i][j]=board[j][i];
                    board[j][i]=temp;
            }


}
void gameover()
{
           glColor3f(0,0,0);
                    glRectf (-30, -20, 70, 10);
}
/*
to calculate score based on the*/
 void   showscore()
{int i,j;
                  float  src=1;
          for( i=0;i<6;i++)
          {         for( j=0;j<6;j++)
                            {
                            if(mark[i][j]==1)
                                    src=src*1.5;
                            }

          }
          totalscore=totalscore+(int)src;
          //printf("ur current score is %d\n",totalscore);

 }
          void deletecol1()
{
int j=0,i=0;
 static int count=0;
for(i=0;i<6;i++)
{
```

```
 for(j=0;j<6;j++)
{

        if(j<5)
{ int flag=0;
        for(int k=0;k<6;k++)
                if(board[k][j]!=3)
                {
                        flag=1;
                }
        if(flag==0)
        { count=count+1;
                for (int k=0;k<6;k++)
                {
                        if(j<5)
                        {
                        board[k][j]=board[k][j+1];
                        board[k][j+1]=3;
                        count--;
                        }
                        else
                        {
                                count--;
                                board[k][j]=3;
                        }
                }

        }

        }
 }
}
//if(count!=0)
//        {
//                deletecol1();
//        }
}




/*
*/

/* the procedure to delete a square*/
void delete1 (void)
{int i,j;
int count[6]={0,0,0,0,0,0};
```

```
for( i=0;i<6;i++)
                for(int j=0;j<6;j++)
                {
                        tempbrd[i][j]=board[i][j];
                        tempmark[i][j]=mark[i][j];
                }

for(j=0;j<6;j++)
        {
                int     stack[7]={0,0,0,0,0,0,0};
                int t=-1;
                for(i=0;i<6;i++)
                {
                        if(mark[i][j]==0)
                        {
                                stack[++t]=board[i][j];
                                count[j]=count[j]+1;
                        }

                }

                for(i=0;i<6;i++)
                {
                        board[i][j]=3;
                }


                for(i=5;i>=5-count[j];i--)
                {

                        if(t>=0)
                        {
                                board[i][j]=stack[t--];

                   }

                }



}
//for(i=0;i<6;i++)
deletecol1();

glutPostRedisplay();
}
```

```
/*
finds the adjecent squares
*/
void FindAjacentSquares(int i,int j,int board[6][6])
{
        if(i>=0 && j>=0&& i<6 && j<6)
        {
                // mark[i][j]=1;
                if(i!=5)
                {
                if(board[i][j]==board[i+1][j])
                {
                        if(mark[i+1][j]!= 1)
                        {
                                mark[i+1][j]=1;
         FindAjacentSquares(i+1,j,board);
              }
                }
                }
                if(j!=5)
                {
                if(board[i][j]==board[i][j+1])
                {
                        if(mark[i][j+1]!=1)
                        {
                        mark[i][j+1]=1;
                        FindAjacentSquares(i,j+1,board);
                }
                        }
                }
                if(i!=0)
                {
                if(board[i][j]==board[i-1][j])
                {
                        if(mark[i-1][j]!=1)
                        {
                        mark[i-1][j]=1;
                        FindAjacentSquares(i-1,j,board);
                   }
                }

                }
                if(j!=0)
                {
```

```
                if(board[i][j]==board[i][j-1])
                {
if(mark[i][j-1]!=1)
{
                        mark[i][j-1]=1;
                        FindAjacentSquares(i,j-1,board);
                }
                }       }
        }
        //glutPostRedisplay();


}
void checkifgameend()
{
        int flag=0;
        int emtpy1=0;
        for(int i=0;i<6;i++)
                for(int j=0;j<6;j++)
                if(board[i][j]!=3)
                        {
                        emtpy1=1;
                        }

                if(emtpy1!=0)
                {
        for(int i=0;i<6;i++)
        for(int j=0;j<6;j++)
        {
                if(j!=5)
                {
                if(board[i][j]==board[i][j+1]&&board[i][j]!=3)
                {flag=1;
                }
                }
                if(i!=5)
                {
                if(board[i][j]==board[i+1][j]&&board[i][j]!=3)
                {
                        flag=1;
                }
                }
                if(j!=0)
                {if(board[i][j]==board[i][j-1]&&board[i][j]!=3)
                {flag=1;
                }
                }
                if(i!=0)
```

```
                {
                if(board[i][j]==board[i-1][j]&&board[i][j]!=3)
                {
                        flag=1;
                }
                }

        }
        if(flag==0)
        {
                glMatrixMode(GL_MODELVIEW);  // Tell opengl that we are doing model
matrix work. (drawing)
     glLoadIdentity(); // Clear the model matrix

                printf("the game is over");
                printf("your total score is %d",totalscore);
end=1;

        }
                }
                else{
                                        printf("the game is over");
                printf("your total score is %d",totalscore);
end=1;

                }
}


/*  processHits prints out the contents of the
 *  selection array.
 */
void processHits (GLint hits, GLuint buffer[])
{

  unsigned int i, j;
  GLuint ii=0, jj=0, names, *ptr;

 // printf ("hits = %d\n", hits);
  ptr = (GLuint *) buffer;
  for (i = 0; i < hits; i++) {     /*  for each hit  */
    names = *ptr;
     ptr++;
 ptr++;ptr++;
   // printf ("   names are ");
    for (j = 0; j < names; j++) { /*  for each name */
//       printf ("%d ", *ptr);
       if (j == 0)  /*  set row and column  */
```

```
          ii = *ptr;
        else if (j == 1)
          jj = *ptr;
        ptr++;
      }
      printf ("\n");

 //     board[ii][jj] = (board[ii][jj] + 1) % 3;
   }
   for(i=0;i<6;i++)
            for(j=0;j<6;j++)
            {
                     mark[i][j]=0;
            }
            if(board[ii][jj]!=3)
            {
   FindAjacentSquares(ii,jj,board);
     showscore();

delete1();
checkifgameend();
            }

//recursive(ii,jj,n);

}


/*  pickSquares() sets up selection mode, name stack,
 *  and projection matrix for picking.  Then the
 *  objects are drawn.
 */
#define BUFSIZE 512

void undo()
{

          for(int i=0;i<6;i++)
                    for(int j=0;j<6;j++)
                    {
                              board[i][j]=tempbrd[i][j];
                              mark[i][j]=tempmark[i][j];
                    }
int j;
                    int  src=1;
          for( i=0;i<6;i++)
          {         for( j=0;j<6;j++)
                              {
```

```
                        if(mark[i][j]==1)
                                src=src*2;
                        }

        }
        if(totalscore>=src)
        {
        totalscore=totalscore-src;
        }
        glutPostRedisplay();
}
void color_menu(int id)
{
//      string s[40];
        switch(id)
        {
        case 1: printf("YOUR SCORE WOULD BE :     %d\n",totalscore);
                        break;
        case 2: undo();
                break;
        case 3: break;
        }
}

void pickSquares(int button, int state, int x, int y)
{
  GLuint selectBuf[BUFSIZE];
  GLint hits;
  GLint viewport[4];
  if(end!=1)
{
if(button==GLUT_RIGHT_BUTTON || state==GLUT_DOWN)
  {

          glutCreateMenu(color_menu);
          glutAddMenuEntry("show score",1);
          glutAddMenuEntry("undo",2);
            glutAddMenuEntry("continue",3);
          glutAttachMenu(GLUT_RIGHT_BUTTON);
          glutPostRedisplay();
  }


  if (button != GLUT_LEFT_BUTTON || state != GLUT_DOWN)
  {
          return;
  }
  glGetIntegerv (GL_VIEWPORT, viewport);
```

```
   glSelectBuffer (BUFSIZE, selectBuf);
   (void) glRenderMode (GL_SELECT);

   glInitNames();
   glPushName(0);

   glMatrixMode (GL_PROJECTION);
   glPushMatrix ();
   glLoadIdentity ();
/*  create 5x5 pixel picking region near cursor location        */
   gluPickMatrix ((GLdouble) x, (GLdouble) (viewport[3] - y),
                  5.0, 5.0, viewport);
   gluOrtho2D (0.0, 6.0, 0.0, 6.0);
   drawSquares (GL_SELECT);
   glMatrixMode (GL_PROJECTION);
   glPopMatrix ();
   glFlush ();
   hits = glRenderMode (GL_RENDER);
   processHits (hits, selectBuf);
   glutPostRedisplay();
}
}
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   drawSquares (GL_RENDER);
   glFlush();
   if(end==1)
{
          glClear(GL_COLOR_BUFFER_BIT);
          glMatrixMode (GL_PROJECTION);  // Tell opengl that we are doing project matrix
work
glLoadIdentity();  // Clear the matrix
glOrtho(-30.0, 70.0, -15.0, 9.0, 0.0, 30.0);   // Setup an Ortho view
glMatrixMode(GL_MODELVIEW);   // Tell opengl that we are doing model matrix work.
(drawing)
glLoadIdentity(); // Clear the model matrix
gameover();
//glDisable(GL_COLOR_MATERIAL);
//glDisable(GL_LIGHTING);
glColor3f(1.0, 0.0, 0.3);
 glClearColor(0.0,0.0,0.0,0.0);
                    Sprint(-30, 7, "GAME");
                    glClearColor(0.0,0.0,0.0,0.0);
                    glColor3f(1.0, 0.3, 0.0);
                    Sprint(-30,4,"OVER");
                    char s[100];
```

```
                    sprintf(s,"%d",totalscore);
//                  Sprint(-30,4,"    ");
       //Sprint(-12,5,s);
                    printf("\n-----------------------------------------------\n");
                    printf("\n-----------------------------------------------\n");
                    printf("\n-----------------------------------------------\n");
                    printf("YOUR SCORE IS      :%d\n",totalscore);
                    // Sprint(-30,5, "game over");
glFlush();
}
}

void reshape(int w, int h)
{
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D (0.0, 6.0, 0.0, 6.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void keyboard(unsigned char key, int x, int y)
{
  switch (key) {
     case 27:
       exit(0);
       break;
  }
}

/* Main Loop */
int main(int argc, char** argv)
{

  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (200, 200);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutReshapeFunc (reshape);
  glutDisplayFunc(display);
  glutMouseFunc (pickSquares);
  glutKeyboardFunc (keyboard);
  glutMainLoop();
  return 0;
}
```

CHAPTER 6

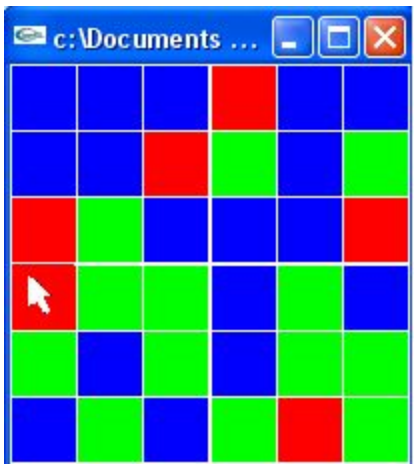
TESTING

Testing is the procedure where the program is executed to check for its proper working.
• The Square Breaker program is tested and verified on Windows Platform.
• The program is tested to verify the positioning and color of the Squares.
• Picking of the squares has been verified. .
• The cascading of the squares has been verified to see to it that they are moving towards their correct position.
•.The end of the game condition has been verified.

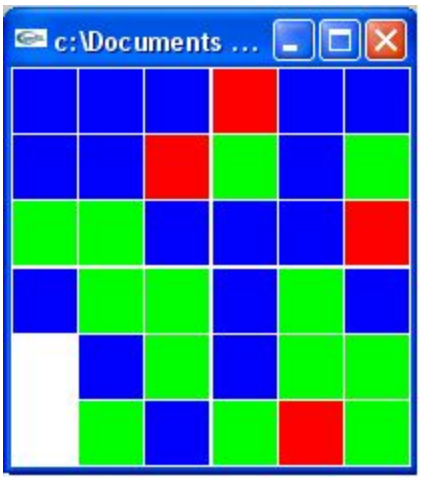
An example of a test case is as given below:
Until a square is selected the matrix remains unchanged .On selecting a square all adjacent squares of the same color are deleted and then cascaded towards the top left corner of the window. The game ends when there are no more adjacent squares of the same color. Under that condition the window displays game over.

CHAPTER 7

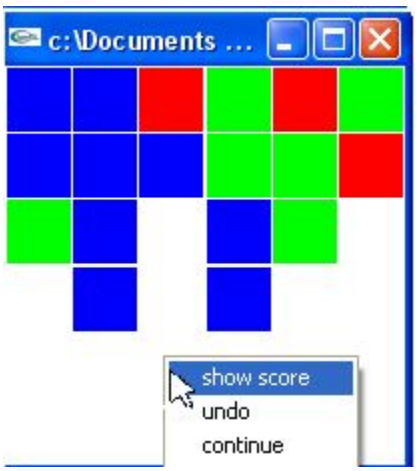
Snapshots



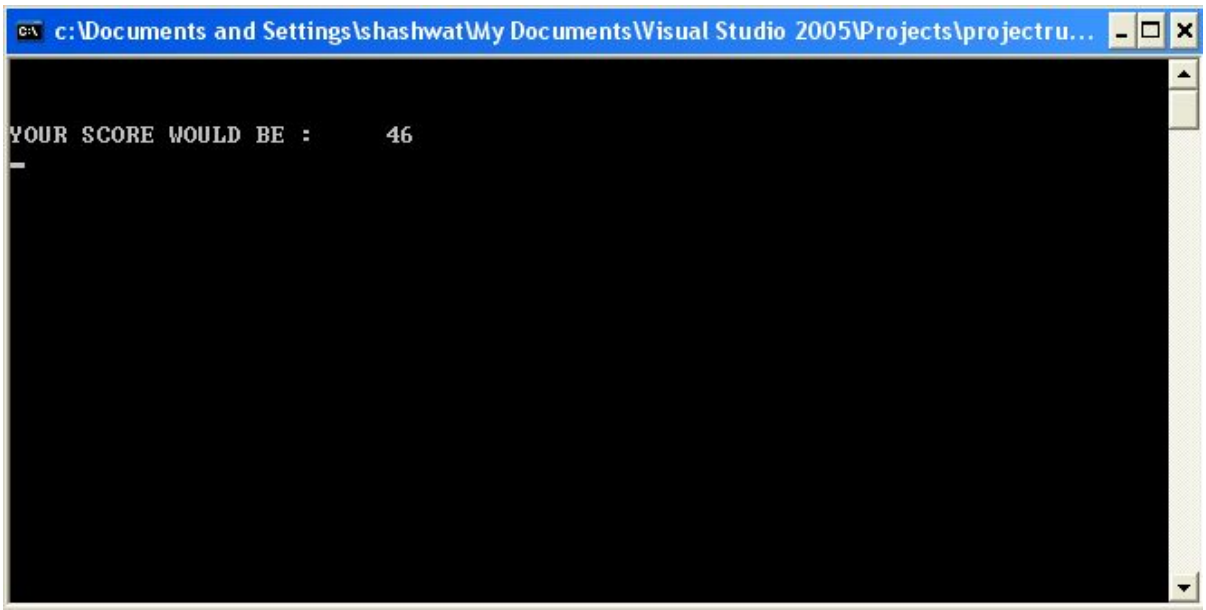
INITIAL MATRIX OF SQUARES



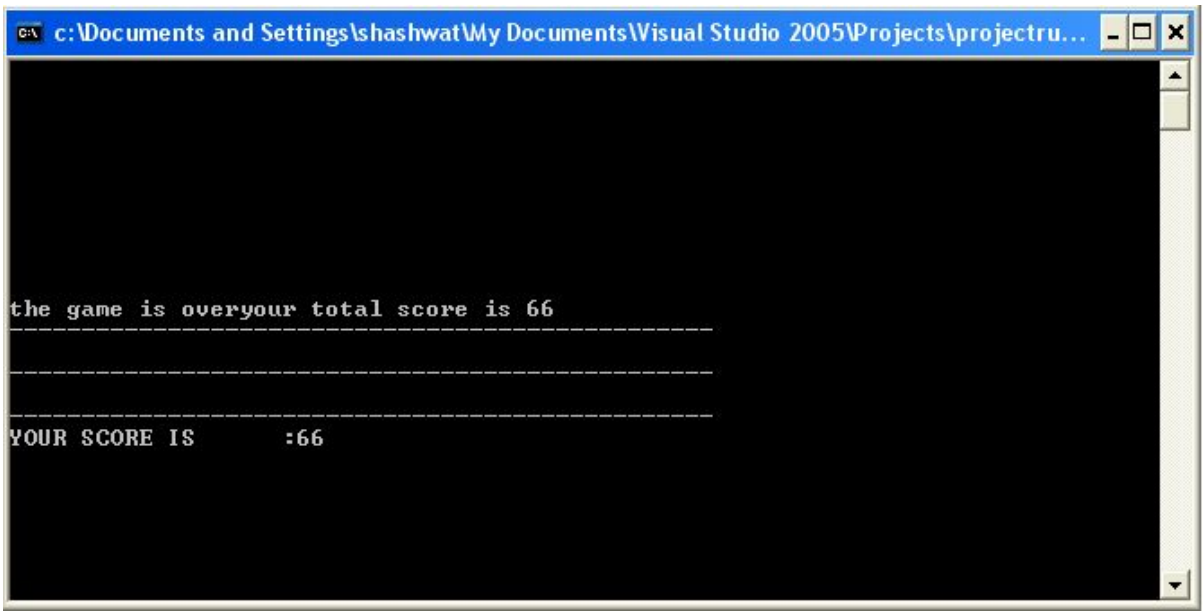
CELL DELETION ON CLICK



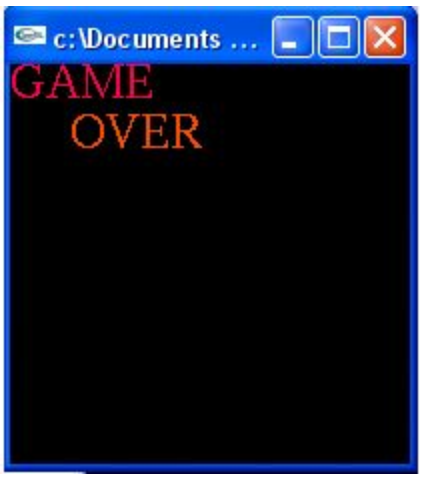
MENU ITEMS



THE RESULT OF SELECTION OF SHOW SCORE OPTION



THE FINAL SCORE DISPLAY ON END OF GAME



END OF GAME WINDOW

Future Enhancements

The future enhancements for this project could include:

1. Using Circles instead of squares

2. Printing score on front window

3. To cascade the squares to bottom-left

Conclusion

The aim of project implement the game Square breaker. The code has been written in OpenGL. The code contains the functions that accomplish all the tasks required for the project. It can be further improved upon to provide better facilities and user interface.

One major improvement which can be made to this package is the inclusion of texture mapping. This package nevertheless provides the basic functionality of square breaker.

Appendix

1. glLoadIdentity : replaces the current matrix with the identity matrix. It is semantically equivalent to calling glLoadMatrix with the identity matrix.

2. glEnable and glDisable enable and disable various capabilities.Use glIsEnabled or glGet to determine the current setting of any capability.

3. glBegin and glEnd delimit the vertices that define a primitive or a group of like primitives.

glBegin accepts a single argument that specifies in which of ten ways the vertices are interpreted.

4. glFlush: Different GL implementations buffer commands in several different locations,including network buffers and the graphics accelerator itself. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

5. glMaterial: glMaterial assigns values to material parameters.

There are two matched sets of material parameters.

One, the front-facing set, is used to shade points, lines, bitmaps and all polygons (when two-sided lighting is disabled), or just front-facing polygons(when two-sided lighting is enabled).

The other set, back-facing, is used to shade back-facing polygons only when two-sided lighting is enabled.

6. glClear: sets the bitplane area of the window to values previously selectedby glClearColor, glClearIndex, glClearDepth,glClearStencil, and glClearAccum.

Multiple color buffers can be cleared simultaneously by selecting more than one buffer at a time using glDrawBuffer.

7. glLight sets the values of individual light source parameters.

light names the light and is a symbolic name of the form GL_LIGHTi,

where i ranges from 0 to the value of GL_MAX_LIGHTS - 1

8. glColorMaterial specifies which material parameters track the current color.

When GL_COLOR_MATERIAL is enabled,the material parameter or parameters specified by mode, of the material or materials specified by face, track the current color at all times.

9. glShadeModel: GL primitives can have either flat or smooth shading. Smooth shading, the default, causes the computed colors of vertices to be interpolated as the primitive is rasterized, typically assigning different colors to each resulting pixel fragment. Flat shading

selects the computed color of just one vertex and assigns it to all the pixel fragments generated by rasterizing a single primitive.

In either case, the computed color of a vertex is the result of lighting if lighting is enabled,or it is the current color at the time the vertex was specified if lighting is disabled.

10. gluPerspective: gluPerspective specifies a viewing frustum into the world coordinate system. In general, the aspect ratio in gluPerspective should match the aspect ratio of the associated viewport. For example, aspect = 2.0 means the viewer's angle of view is twice as wide in x as it is in y. If the viewport is twice as wide as it is tall, it displays the image without distortion

11. glTranslate — multiply the current matrix by a translation matrix

12. glRotate — multiply the current matrix by a rotation matrix

13 .glPushMatrix and glPopMatrix— push and pop the current matrix stack

14. glColor — set the current color

15. glutAddMenuEntry - adds a menu entry to the bottom of the current menu.

16.glutSwapBuffers - swaps the buffers of the current window if double buffered.

17.glutSolidSphere - render a solid or wireframe sphere respectively.

18.glutMainLoop - enters the GLUT event processing loop.

19.glutInitDisplayMode - sets the initial display mode.

20.glutInit - initialize the GLUT library.

21.glMatrixMode — specify which matrix is the current matrix

22.glClearDepth — specify the clear value for the depth buffer

Bibliography

Books:

- **Edward Angel**, "Interactive Computer Graphics",5th edition,Pearson Education,2005
- **Donald D Hearn** and **M. Pauline Baker,** " Computer Graphics with OpenGL", 3rd Edition

Websites:

- **http://www.opengl.org**
- **http://glprogramming.com/red**
- **http://jerome.jouvie.free.fr/OpenGl**