Computer Graphics (UCS505)

Project on

Multi-Level Shape Shooter Archery Game

Submitted By

Ekaspreet Kaur

Kriti Singhal

Group No. 4

B.E. Third Year – CSE

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

Table of Contents

INTRODUCTION

Computer Graphics is a powerful tool for the rapid and economical production of pictures. There is virtually no area in which Graphical displays cannot be used to some advantage, so it is not surprising to find the use of CG so widespread. Nowadays Computer Graphics are used in almost all areas ranging from science, engineering, medicine, business, industry, government, art, entertainment, education, and training.

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms.

Multi-Level shape shooter archery game

This project is a twist on the classic archery game where user has to complete 3 different levels. To make this project more interesting, the number of maximum moves for each level are fixed. Winning the game is only possible if you use the most optimized moves. This project is made using OpenGL which includes all the basic library functions. There are infinite invisible arrows and a moving platform to shoots, through which the target block is destroyed.

The main objectives of our project are:

- Showcase the use of OpenGL to make interactive user game
- Illustrating the keyboard interaction.
- To provide entertainment to the user

Rules and Working

This is an archery game built using OPEN GL. When the program is executed the front sheet of the project is displayed. Then the player must press the ‘t’ button to start the game. The game screen is displayed where initially the rules are displayed through which the user understands the game and then ‘m’ needs to pressed to start the game when the game starts the user needs to clear three levels in order to complete the game throughout which he has infinite invisible arrows but a limited movement.

Game Instructions

- The player should press the ‘m’ key to start the game.
- There are 3 Levels in the game.
- The player should destroy the targets in order to win the levels.
- The player should press 'w' to shoot the arrow which is invisible.
- The player should press 'a' to move the platform left.
- The player should press 'd' to move the platform right.
- There is a restriction on the number of platform movements that can be performed.
- The targets must be destroyed within the total number of movements provided.
- Every level has a different type of target design.
- There are twelve positions to which the platform can move to.

COMPUTER GRAPHICS CONCEPTS USED

Pixel operation

First, data in the form of pixels is taken from system memory. These pixels may be in different formats, so they are converted into the right number of components. Then, the data is adjusted in terms of size, bias, and processed using a pixel map. The results are then limited to a certain range and either stored in texture memory or sent to the Rasterization step. If data is taken from the frame buffer, operations like scaling, biasing, mapping, and clamping are performed on the pixels. After that, the results are converted into an appropriate format and returned to system memory. There are also special operations to copy data from one part of the frame buffer to another or to the texture memory. All these operations are performed in one go before the data is written to texture memory or back to the frame buffer.

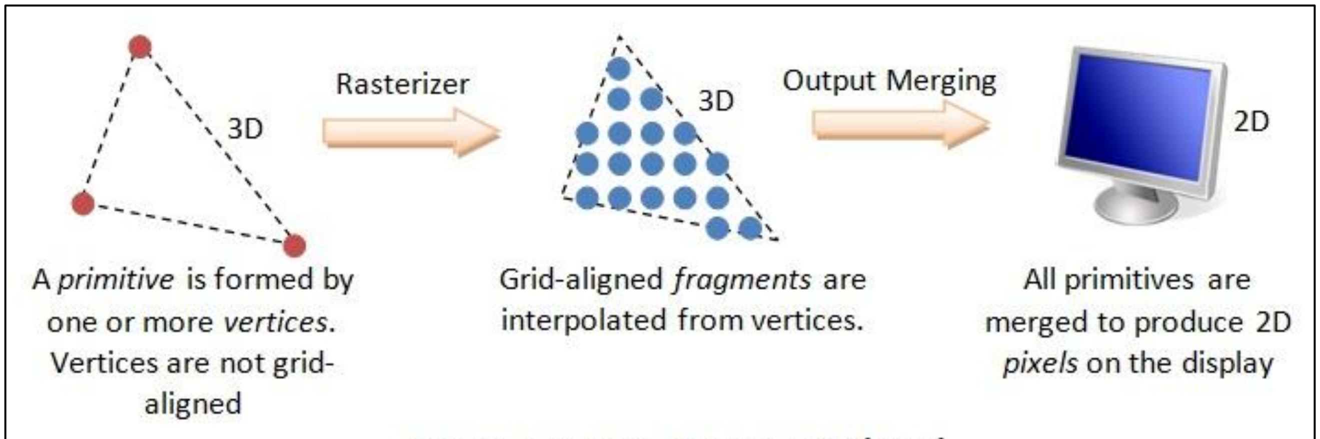
All data, whether it describes geometry or pixels, can be saved in a *display list* for current or later use. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

Transforming to Window Coordinates

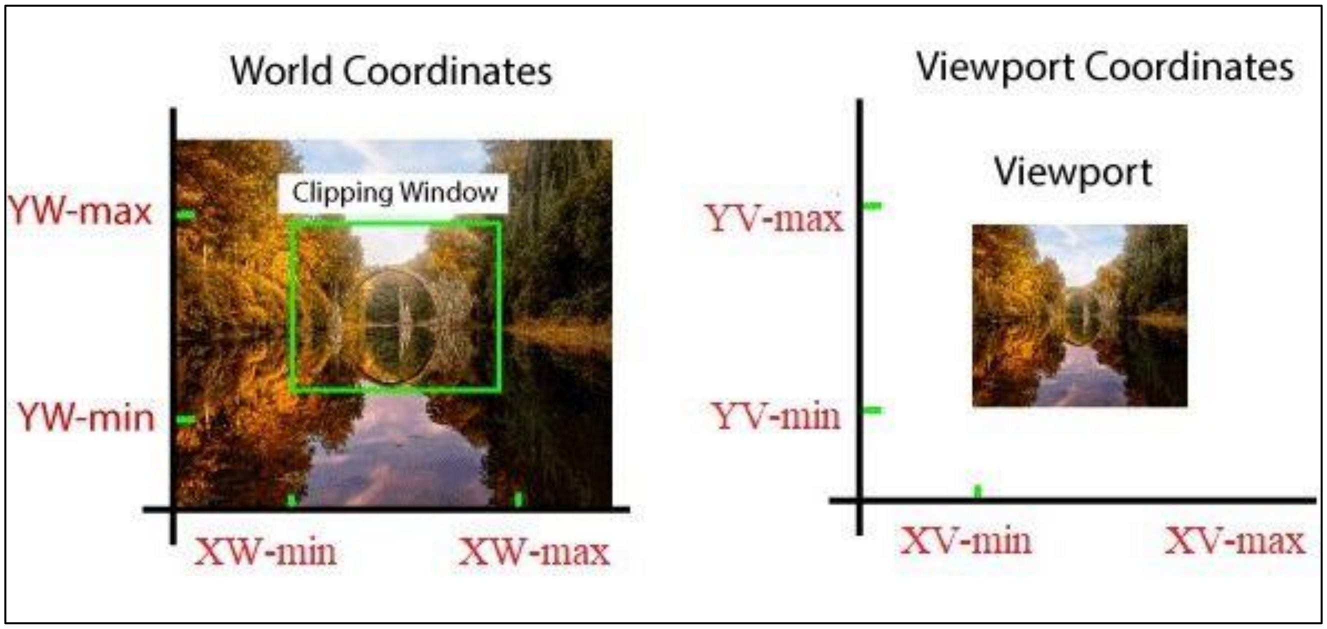
Before clip coordinates can be converted to window coordinates, they are normalized by dividing by the value of w to yield normalized device coordinates. After that, the viewport transformation applied to these normalized coordinates produces window coordinates. One can control the viewport, which determines the area of the on-screen window that displays an image, with glDepthRange () and glViewport ().

Matrix Transformations

Vertices and normals are transformed by the model-view and projection matrices before they're used to produce an image in the frame buffer. You can use commands such as glMatrixMode (), glMultMatrix (), glRotate (), glTranslate (), and glScale () to compose the desired transformations, or you can directly specify matrices with glLoadMatrix () and glLoadIdentity (). Use glPushMatrix () and glPopMatrix () to save and restore model view and projection matrices on their respective stacks.


3D
Rasterizer
3D
Output Merging
2D
A *primitive* is formed by one or more *vertices*. Vertices are not grid-aligned
Grid-aligned *fragments* are interpolated from vertices.
All primitives are merged to produce 2D *pixels* on the display

Vertex, Primitives, Fragment and Pixel



Functions Used

The whole program has been implemented in the C++ language. The bottom line of the design is keyboard interaction, and some functions are used to print the text on the screen

4.1 Built-in functions

- **glutCreateWindow (char* name)**

The glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name. Implicitly, the current window is set to the newly-created window. Each created window has a unique associated OpenGL context. name - ASCII character string for use as window name.

- **glutDestroyWindow (int win)**

This function destroys the specified window. glutDestroyWindow destroys the window specified by the win and the window's associated OpenGL context, logical colormap (if the window is color index), and overlay and related state (if an overlay has been established). Any sub windows of destroyed windows are also destroyed by glutDestroyWindow. If the win was the current window, the current window becomes invalid (glutGetWindow will return zero).

win – identifies the GLUT window to destroy.

- **glutInit (int argc, char** argv)**

glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command-line options.

argc - A pointer to the program's unmodified argc variable from main.

argv - The program's unmodified argv variable from main. Like argc, the data for argv will be updated because glutInit extracts any command-line options understood by the GLUT library.

- **glutInitDisplayMode (unsigned int mode)**

glutInitDisplayMode sets the initial display mode. The initial display mode is used when creating top-level windows, sub windows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.

Mode - Display mode, normally the bitwise OR-ing of GLUT display mode bit masks. It can have values like the following:

GLUT_SINGLE - Bit mask to select a single buffered window. GLUT_DOUBLE - Bit mask to select a double-buffered window. GLUT_DEPTH - Bit mask to select a window with a depth buffer. GLUT_RGB - Bit mask to select an RGB mode window.

GLUT_INDEX - Bit mask to select a color index mode window. GLUT_STEREO - Bit mask to select a stereo window.

- **glClearColor()**

  glClearColor specifies the red, green, blue, and alpha values used by glClear to clear the color buffers. Values specified by glClearColor are clamped to the range [0,1]. It can have parameters like red, green, blue, and alpha. Specify the red, green, blue, and alpha values used when the color buffers are cleared. The initial values are all 0.

- **glutDisplayFunc(void (*func)(void))**

  glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the

  display callback for the window is called. Before the callback, the current window is set to the window needing to be redisplayed and the layer in use is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback.

  func - The new display callback function.

- **glutPostRedisplay(void)**

  Mark the normal plane of current window as needing to be redisplayed. The next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback. glutPostRedisplay may be called within a window's display or overlay display callback to re-mark that window for redisplay.

- **glutKeyboardFunc(void (*func)(unsigned char key, int x, int y))**

  glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII keystrokes in the window are ignored. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks.

  func – The new keyboard callback function.

- **glutMainLoop(void)**

  glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

- **glutFullScreen(void)**

  glutFullScreen requests that the current window be made full screen. The exact semantics of what full screen means may vary by window system. The intent is to make the window as large as possible and disable any window decorations or borders added the window system. The window width and height are not guaranteed to be the same as the screen width and height, but that is the intent of making a window full screen. glutFullScreen is defined to work only on top-level windows.

- **glClear(GLbitfield mask)**

  glClear sets the bitplane area of the window to values previously selected by glClearColor, glClearIndex, glClearDepth, glClearStencil, and glClearAccum. Multiple color buffers can be cleared simultaneously by selecting more than one buffer at a time using glDrawBuffer. glClear takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared.

  mask - Bitwise OR of masks that indicate the buffers to be cleared. The four masks are GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT, GL_ACCUM_BU FFER_BIT, and GL_STENCIL_BUFFER_BIT.

  The values for glClear are:

  GL_COLOR_BUFFER_BIT - Indicates the buffers currently enabled for color writing.

  GL_DEPTH_BUFFER_BIT - Indicates the depth buffer. GL_ACCUM_BUFFER_BIT - Indicates the accumulation buffer. GL_STENCIL_BUFFER_BIT - Indicates the stencil buffer.

- **glShadeModel(GLenum mode)**

  glShadeModel 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 the lighting is enabled, or it is the current color at the time the vertex was specified if the lighting is disabled.

mode - Specifies a symbolic value representing a shading technique. Accepted values are GL_FLAT and GL_SMOOTH. The initial value is GL_SMOOTH.

- **glOrtho()**

glOrtho multiplies the current matrix with an orthographic matrix. glOrtho describes a transformation that produces a parallel projection. The current matrix is multiplied by this matrix and the result replaces the current matrix. It can have parameters like:

left, right - Specify the coordinates for the left and right vertical clipping planes. bottom, top -Specify the coordinates for the bottom and top horizontal clipping planes. nearVal, farVal - Specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.

- **glSwapBuffers(void)**

Performs a buffer swap on the layer in use for the current window. Specifically, glutSwapBuffers promotes the contents of the back buffer of the layer in use of the current window to become the contents of the front buffer. The contents of the back buffer then become undefined. The update typically takes place during the vertical retrace of the monitor, rather than immediately after glutSwapBuffers is called. An implicit glFlush is done by glutSwapBuffers before it returns. Subsequent OpenGL commands can be issued immediately after calling glutSwapBuffers but are not executed until the buffer exchange is completed. If the layer in use is not double buffered, glutSwapBuffers have no effect.

- **glFlush(void)**

  This function force execution of GL commands in finite time. Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. glFlush empties all 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, it does complete in finite time. Because any GL program might be executed over a network, or on an accelerator that buffers commands, all programs should call glFlush whenever they count on having all their previously issued commands completed. For example, call glFlush before waiting for user input that depends on the generated image.

- **glPushMatrix(void)**

  glPushMatrix pushes the current matrix stack down by one, duplicating the current matrix. That is, after a glPushMatrix call, the matrix on top of the stack is identical to the one below it.

- **glPopMatrix(void)**

  glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack.

- **glutMainLoop(void)**

  glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any call-backs that have been registered.

- **glutInitWindowPosition(int x, int y)**

  glutInitWindowPosition set the initial window position. The initial value of the initial window position GLUT state is -1 and -1. If either the X or Y component to the initial window position is negative, the actual window position is left to the window system to determine. The intent of the initial window position values is to provide a suggestion to the window system for a window's initial position. The window system is not obligated to use this information. Therefore, GLUT programs should not assume the window was created at the specified position. A GLUT program should use the window's reshape callback to determine the true size of the window.

  x – Window X location in pixels.

  y – Window Y location in pixels.

- **glutInitWindowSize(int width, int height)**

  The initial value of the initial window size GLUT state is 300 by 300. The initial window size components must be greater than zero. The intent of the initial window position values is to provide a suggestion to the window system for a window's initial

  size. The window system is not obligated to use this information. Therefore, GLUT programs should not assume the window was created at the specified size. A GLUT program should use the window's reshape callback to determine the true size of the window.

  width – Width in pixels.

  height – Height in pixels.

- **glRectf(x1, y1, x2, y2)**

glRect supports efficient specification of rectangles as two corner points. Each rectangle command takes four arguments, organized either as two consecutive pairs of x y coordinates or as two pointers to arrays, each containing an x y pair. The resulting rectangle is defined in the z = 0 plane.

x1, y1 - Specify one vertex of a rectangle.

x2, y2 - Specify the opposite vertex of a rectangle.

4.2 User-define function

- **void target()**

This function is used to render the targets present in the game according to the level the user is playing it is also made so that after the user has destroyed the specific target it is not to be displayed the function uses ‘countXX’ of integer type to check if the target blocks need to displayed or not.

- **void frontsheet()**

This function is used to write the contents of the front sheet. glClearColor function is used to give the color to the front sheet. glColor3f is used to give color to the strings.

- **void platform(int PlatformPosition)**

This function is used to draw the platform and control its movement accordingly it is written in a way such that when the keys to move the platform(‘a’ or ‘d’) are clicked, it moves the platform from the present location to required location.

- **void platform()**

  This function is used to check if the game has been won. It uses ‘winX’ a variable of integer type to check if all the targets have been destroyed under the given moves and it they have it displays that the level has been cleared.

- **void loss()**

  This function is used to check if the user overuses the movement chance(moves) if so then this function displays the ‘You lost level’ message

- **void keys(unsigned char key, int x, int y)**

  This function is used to assign keys to the different operations

  ‘t’ is used to move from front page to actual game

  ‘m’ is used to start the game and move to level 1

  ‘n’ is assigned to level 2

  ‘b’ is assigned to level 3

  ‘a’ and ‘d’ are used to move the platform

  ‘w’ is used to fire the invisible arrow

- **void alwaysDisplay()**

  This function is used to display the rules at the beginning of the game and then display the total moves and which level the user is playing and draw the border to the game

- **void print_text(int x, int y, char str[], float r, float g, float b)**

  These functions are used to display the front page onto the screen.

CODE

```
#include <iostream>
#include <math.h>
#include <glut.h>
#include <string.h>
#include<stdlib.h>

int count1 = 0, count2 = 0, count3 = 0, count4 = 0, count5 = 0, count6 = 0, count7 = 0,
count8 = 0; //countX level 2 targets

int count21 = 0, count22 = 0, count23 = 0, count24 = 0, count25 = 0; //count2X level 1
targets

int count31 = 0, count32 = 0, count33 = 0, count34 = 0, count35 = 0, count36 = 0;
//count3X level 3 targets

int pm1 = 0, pm2 = 0, pm3 = 0;//platform movement increase to decide win or not

int gs = 0; //game start to display m-> new level
//'ww' -> 'w' key working boolean
int ww1 = 0; //should w work for level one
int ww2 = 0;  //should w work for level tho
int ww3 = 0;  //should w work for level three
int pp = 6;   //platform position
int win = 0;  //win
int win2 = 0;  //win level 2
int win3 = 0;  //win level 3
int lev = 0;  //level 1 2 3
int fs = 1; //front screen
char dash[200] = "--------------------------------------------------------";
char college[100] = "Thapar institute of engineering and technology";
char dept[100] = "COMPUTER SCIENCE & ENGINNERING";
char heading[100] = "SUBMITTED BY";
char name1[100] = "Ekaspreet kaur                    102017078";
char name2[100] = "Kriti Singhal                     102017079";

char emsg[100] = "Press 't' to start the game";
char title[50] = "Shape Shooter Archery Game";


void print_text(int x, int y, char str[], float r, float g, float b) {
   glRasterPos2i(x, y);                                        //displays msg passed as
parameter
   for (int i = 0; str[i] != '\0'; i++) {
      glColor3f(r, g, b);
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, str[i]);
   }
}
void display_string(int x, int y, char* string, int font) {      //display the string passed as
parameter
   int len, i;
   glColor3f(0.257, 0.446, 74.399);    //All the blue color of the game interface
   glRasterPos2f(x, y);
   len = (int)strlen(string);
   for (i = 0; i < len; i++) {
      if (font == 1)
         glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, string[i]);
      if (font == 2)
         glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, string[i]);
      if (font == 3)
         glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, string[i]);
```

```
        if (font == 4)
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_10, string[i]);
    }
}

void reshape(int w, int h) {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, w, 0.0, h);
    glViewport(0.0, 0.0, w, h);
}
void target() {
    if (lev == 1) {
        if (count1 == 1) {
            glColor3f(1, 0, 0);
            glPointSize(30);
            glBegin(GL_POINTS);
            glVertex2d(125, 225);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glVertex2d(100, 200);
            glVertex2d(150, 200);
            glVertex2d(150, 250);
            glVertex2d(100, 250);
            glEnd();
        }

        if (count2 == 1) {
            glColor3f(1, 0, 0);
            glPointSize(30);
            glBegin(GL_POINTS);
            glVertex2d(125, 375);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glVertex2d(100, 350);
            glVertex2d(150, 350);
            glVertex2d(150, 400);
            glVertex2d(100, 400);
            glEnd();

        }
        if (count3 == 1) {
            glColor3f(1, 0, 0);
            glPointSize(30);
            glBegin(GL_POINTS);
            glVertex2d(175, 525);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glVertex2d(150, 500);
            glVertex2d(200, 500);
            glVertex2d(200, 550);
            glVertex2d(150, 550);
            glEnd();

        }if (count4 == 1) {
            glColor3f(1, 0, 0);
            glPointSize(30);
            glBegin(GL_POINTS);
            glVertex2d(275, 325);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glVertex2d(250, 300);
            glVertex2d(300, 300);
            glVertex2d(300, 350);
            glVertex2d(250, 350);
            glEnd();

        }
        if (count5 == 1) {
            glColor3f(1, 0, 0);
            glPointSize(30);
            glBegin(GL_POINTS);
            glVertex2d(325, 175);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glVertex2d(300, 150);
            glVertex2d(350, 150);
            glVertex2d(350, 200);
            glVertex2d(300, 200);
            glEnd();

        }
        if (count6 == 1) {
            glColor3f(1, 0, 0);
            glPointSize(30);
            glBegin(GL_POINTS);
            glVertex2d(375, 425);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glVertex2d(350, 400);
            glVertex2d(400, 400);
            glVertex2d(400, 450);
            glVertex2d(350, 450);
            glEnd();

        }
        if (count7 == 1) {
            glColor3f(1, 0, 0);
            glPointSize(30);
            glBegin(GL_POINTS);
            glVertex2d(425, 225);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glVertex2d(400, 200);
            glVertex2d(450, 200);
            glVertex2d(450, 250);
            glVertex2d(400, 250);
            glEnd();

        }
        if (count8 == 1) {
            glColor3f(1, 0, 0);
            glPointSize(30);
            glBegin(GL_POINTS);
            glVertex2d(475, 525);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glVertex2d(450, 500);
            glVertex2d(500, 500);
            glVertex2d(500, 550);
            glVertex2d(450, 550);
            glEnd();
        }
```

```
    glutPostRedisplay();
}

if (lev == 2) {
    if (count21 == 1) {
        glColor3f(0, 0, 1);
        glBegin(GL_POLYGON);
        glVertex2d(150, 200);
        glVertex2d(200, 200);
        glVertex2d(175, 250);
        glEnd();
    }
    if (count22 == 1) {
        glColor3f(0, 0, 1);
        glPointSize(30);
        glBegin(GL_POLYGON);
        glVertex2d(50, 350);
        glVertex2d(100, 350);
        glVertex2d(75, 400);
        glEnd();

    }
    if (count23 == 1) {
        glColor3f(0, 0, 1);
        glBegin(GL_POLYGON);
        glVertex2d(250, 500);
        glVertex2d(300, 500);
        glVertex2d(275, 550);
        glEnd();
    }
```

```
        if (count24 == 1) {
            glColor3f(0, 0, 1);
            glBegin(GL_POLYGON);
            glVertex2d(400, 400);
            glVertex2d(450, 400);
            glVertex2d(425, 450);
            glEnd();

        }
        if (count25 == 1) {
            glColor3f(0, 0, 1);
            glBegin(GL_POLYGON);
            glVertex2d(300, 250);
            glVertex2d(350, 250);
            glVertex2d(325, 300);
            glEnd();
        }
        glutPostRedisplay();
    }

    if (lev == 3) {
        if (count31 == 1) {
            glColor3f(0, 0, 0);
            glPointSize(15);
            glBegin(GL_POINTS);
            glVertex2d(125, 525);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glVertex2d(100, 525);   //+25y
            glVertex2d(125, 550);    //-25x+50y
            glVertex2d(150, 525);    //-25y
            glVertex2d(125, 500);    //+25x-50y
            glEnd();
        }
        if (count32 == 1) {
            glColor3f(0, 0, 0);
            glPointSize(15);
            glBegin(GL_POINTS);
            glVertex2d(175, 375);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glVertex2d(150, 375);
            glVertex2d(175, 400);
            glVertex2d(200, 375);
            glVertex2d(175, 350);
            glEnd();

        }
        if (count33 == 1) {
            glColor3f(0, 0, 0);
            glPointSize(15);
            glBegin(GL_POINTS);
            glVertex2d(425, 425);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glVertex2d(400, 425);
            glVertex2d(425, 450);
            glVertex2d(450, 425);
            glVertex2d(425, 400);
            glEnd();

        }
```

```
if (count34 == 1) {
    glColor3f(0, 0, 0);
    glPointSize(15);
    glBegin(GL_POINTS);
    glVertex2d(325, 375);
    glEnd();
    glBegin(GL_LINE_LOOP);
    glVertex2d(300, 375);
    glVertex2d(325, 400);
    glVertex2d(350, 375);
    glVertex2d(325, 350);
    glEnd();

}
if (count35 == 1) {
    glColor3f(0, 0, 0);
    glPointSize(15);
    glBegin(GL_POINTS);
    glVertex2d(375, 275);
    glEnd();
    glBegin(GL_LINE_LOOP);
    glVertex2d(350, 275);
    glVertex2d(375, 300);
    glVertex2d(400, 275);
    glVertex2d(375, 250);
    glEnd();

}
if (count36 == 1) {
    glColor3f(0, 0, 0);
    glPointSize(15);
```

```
            glBegin(GL_POINTS);
            glVertex2d(475, 525);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glVertex2d(450, 525);
            glVertex2d(475, 550);
            glVertex2d(500, 525);
            glVertex2d(475, 500);
            glEnd();

        }
        glutPostRedisplay();
    }
}

void platform(int pp) {
    if (pp == 1) {
        glColor3f(0, 1, 0);
        glBegin(GL_POLYGON);
        glVertex2i(0, 50);
        glVertex2i(50, 50);
        glVertex2i(50, 80);
        glVertex2i(0, 80);
        glEnd();
    }
    if (pp == 2) {
        glColor3f(0, 1, 0);
        glBegin(GL_POLYGON);
        glVertex2i(50, 50);
        glVertex2i(100, 50);
        glVertex2i(100, 80);
    glVertex2i(50, 80);
    glEnd();
}
if (pp == 3) {
    glColor3f(0, 1, 0);
    glBegin(GL_POLYGON);
    glVertex2i(100, 50);
    glVertex2i(150, 50);
    glVertex2i(150, 80);
    glVertex2i(100, 80);
    glEnd();
}
if (pp == 4) {
    glColor3f(0, 1, 0);
    glBegin(GL_POLYGON);
    glVertex2i(150, 50);
    glVertex2i(200, 50);
    glVertex2i(200, 80);
    glVertex2i(150, 80);
    glEnd();
}
if (pp == 5) {
    glColor3f(0, 1, 0);
    glBegin(GL_POLYGON);
    glVertex2i(200, 50);
    glVertex2i(250, 50);
    glVertex2i(250, 80);
    glVertex2i(200, 80);
    glEnd();
}
```

```
if (pp == 6) {
   glColor3f(0, 1, 0);
   glBegin(GL_POLYGON);
   glVertex2i(250, 50);
   glVertex2i(300, 50);
   glVertex2i(300, 80);
   glVertex2i(250, 80);
   glEnd();
}
if (pp == 7) {
   glColor3f(0, 1, 0);
   glBegin(GL_POLYGON);
   glVertex2i(300, 50);
   glVertex2i(350, 50);
   glVertex2i(350, 80);
   glVertex2i(300, 80);
   glEnd();
}
if (pp == 8) {
   glColor3f(0, 1, 0);
   glBegin(GL_POLYGON);
   glVertex2i(350, 50);
   glVertex2i(400, 50);
   glVertex2i(400, 80);
   glVertex2i(350, 80);
   glEnd();
}
if (pp == 9) {
   glColor3f(0, 1, 0);
   glBegin(GL_POLYGON);
```

```
    glVertex2i(400, 50);
    glVertex2i(450, 50);
    glVertex2i(450, 80);
    glVertex2i(400, 80);
    glEnd();
  }
  if (pp == 10) {
    glColor3f(0, 1, 0);
    glBegin(GL_POLYGON);
    glVertex2i(450, 50);
    glVertex2i(500, 50);
    glVertex2i(500, 80);
    glVertex2i(450, 80);
    glEnd();
  }
  if (pp == 11) {
    glColor3f(0, 1, 0);
    glBegin(GL_POLYGON);
    glVertex2i(500, 50);
    glVertex2i(550, 50);
    glVertex2i(550, 80);
    glVertex2i(500, 80);
    glEnd();
  }
  if (pp == 12) {
    glColor3f(0, 1, 0);
    glBegin(GL_POLYGON);
    glVertex2i(550, 50);
    glVertex2i(600, 50);
    glVertex2i(600, 80);
    glVertex2i(550, 80);
```

```
        glEnd();
    }
}

void winF() {
    if (win == 8) {
        char ch[120] = "YOU CLEARED LEVEL 2";
        display_string(20, 300, ch, 1);
        pm1 = 0;
        //platform movement counter of level 1 is set to zero when won
        char ch1[120] = "PRESS n TO PLAY AGAIN";
        display_string(20, 100, ch1, 2);
        char ch2[120] = "PRESS b TO PLAY NEXT LEVEL";
        display_string(20, 80, ch2, 2);
        char chk[100] = "PRESS q TO EXIT";
        display_string(20, 60, chk, 2);
        glutIdleFunc(NULL);
    }
    if (win2 == 5) {
        char j[200] = "YOU CLEARED LEVEL 1!";
        display_string(20, 300, j, 1);
        pm2 = 0;
        //platform movement counter of level 2 is set to zero when won
        char k[200] = "PRESS m TO PLAY AGAIN";
        display_string(20, 100, k, 2);
        char km[200] = "PRESS n TO PLAY NEXT LEVEL";
        display_string(20, 80, km, 2);
        char kk[200] = "PRESS q TO EXIT";
        display_string(20, 60, kk, 2);
        glutIdleFunc(NULL);
    }
```

```
  if (win3 == 6) {
    char s[200] = "YOU CLEARED LEVEL 3!";
    display_string(20, 300, s, 1);
    char lk[200] = "YOU CLEARED THE GAME!";
    display_string(20, 270, lk, 1);
    pm3 = 0;
    char hb[200] = "PRESS b TO PLAY AGAIN";//platform movement counter of level 3
is set to zero when won
    display_string(20, 100, hb, 2);
    char lss[200] = "PRESS q TO EXIT";
    display_string(20, 80, lss, 2);
    glutIdleFunc(NULL);
  }
}
void loss() {
  if (pm1 > 10) {
    char ps[200] = "YOU LOST LEVEL 2!!!!(you ran out of moves)";
    display_string(20, 300, ps, 1);
    char kkm[200] = "PRESS n TO PLAY AGAIN";
    display_string(20, 100, kkm, 2);
    char mxn[200] = "PRESS q TO EXIT";
    display_string(20, 80, mxn, 2);
    //int count1=0,count2=0,count3=0,count4=0,count5=0,count6=0,count7=0,count8=0;
Not required
    ww1 = 1; //make the w key unusable
    glutPostRedisplay();
    glutIdleFunc(NULL);
  }
  if (pm2 > 11) {
    char ksn[200] = "YOU LOST LEVEL 1!!!!(you ran out of moves)";
    display_string(20, 300, ksn, 1);
    char kxb[200] = "PRESS m TO PLAY AGAIN";
```

```
        display_string(20, 100, kxb, 2);
        char muey[200] = "PRESS q TO EXIT";
        display_string(20, 80, muey, 2);
        //int count21=0,count22=0,count23=0,count24=0,count25=0;   Not required
        ww2 = 1;  //make the w key unusable
        glutPostRedisplay();
        glutIdleFunc(NULL);
    }

    if (pm3 > 11) {
        char kss[200] = "YOU LOST LEVEL 3!!!!(you ran out of moves)";
        display_string(20, 300, kss, 1);
        char nbx[200] = "PRESS b TO PLAY AGAIN";
        display_string(20, 100, nbx, 2);
        char nxb[200] = "PRESS q TO EXIT";
        display_string(20, 80, nxb, 2);
        ww3 = 1;   //make the w key unusable
        //int count31=0,count32=0,count33=0,count34=0,count35=0,count36=0;     Not
required
        glutPostRedisplay();
        glutIdleFunc(NULL);
    }
}
void alwaysDisplay() {
    if (gs == 0) {
        char n8[200] = "Rules:";
        display_string(100, 520, n8, 1);
        char n0[200] = ">>There are 3 Levels in the game";
        display_string(100, 490, n0, 1);
        char m2[200] = ">>Press 'w' to shoot the arrow which is invisible";
        display_string(100, 470, m2, 1);
        char m9[200] = ">>Press 'a' to move the platform left";
        display_string(100, 450, m9, 1);
        char m27[200] = ">>Press 'd' to move the platform right";
        display_string(100, 430, m27, 1);
        char m77[200] = "The twist in the game is:";
        display_string(100, 410, m77, 1);
        char m22[200] = ">>There are infinite invisible arrows";
        display_string(100, 390, m22, 1);
        char m21[200] = ">>The challenge is to destroy the targets ";
        display_string(100, 370, m21, 1);



        char m210[200] = "within given moves";
        display_string(100, 350, m210, 1);
        char ni[200] = "PRESS m TO START NEW GAME";
        display_string(150, 300, ni, 1);
    }

    if (lev == 2) {
        char jj[200] = "LEVEL 1";
        display_string(250, 600, jj, 1);
        char n[200] = "Total Moves:11";
        display_string(480, 600, n, 2);
    }
    if (lev == 1) {
        char kkh[200] = "LEVEL 2";
        display_string(250, 600, kkh, 1);
        char lm[200] = "Total Moves:10";
        display_string(480, 600, lm, 2);
    }
```

```
    if (lev == 3) {
        char mx[200] = "LEVEL 3";
        display_string(250, 600, mx, 1);
        char x9[200] = "Total Moves:11";
        display_string(480, 600, x9, 2);
    }
    //code for border 2 line loops
    glColor3b(1, 1, 1);
    glBegin(GL_LINE_LOOP);
    glLineWidth(5);
    glVertex2d(10, 10);
    glVertex2d(10, 635);
    glVertex2d(640, 635);
    glVertex2d(640, 10);
    glVertex2d(10, 10);

    glEnd();
    glColor3b(0, 1, 0);
    glBegin(GL_LINE_LOOP);
    glLineWidth(5);
    glVertex2d(15, 15);
    glVertex2d(15, 630);
    glVertex2d(635, 630);
    glVertex2d(635, 15);
    glVertex2d(15, 15);
    glEnd();
}

void frontscreen()
{
    glClear(GL_COLOR_BUFFER_BIT);
    print_text(10, 230, dash, 0.0, 0.0, 0.0);//displays dashes
    print_text(10, 230, dash, 0.0, 0.0, 0.0);
    print_text(10, 190, dash, 0.0, 0.0, 0.0);
    print_text(110, 540, college, 0.0, 0.0, 1.0);            //displays college name
    print_text(120, 480, dept, 0.7, 0.0, 1.0);               //displays dept.
    print_text(180, 410, title, 0.0, 0.0, 0.0);           //displays project name
    print_text(10, 375, dash, 0.0, 0.0, 0.0);             //displays dashes
    print_text(100, 355, heading, 0.0, 0.0, 0.0);
    print_text(10, 340, dash, 0.0, 0.0, 0.0);//displays heading of table
    print_text(100, 320, name1, 0.0, 0.0, 0.0);          //displays  name1 in table
    print_text(100, 295, name2, 0.0, 0.0, 0.0);          //displays name2 in table
    print_text(170, 80, emsg, 1.0, 0.1, 1.0);                //displays "enter msg"
    glFlush();
}


void display(){
    glClear(GL_COLOR_BUFFER_BIT);
    if (fs == 1) {              //fs-> front screen if it is 1 then the project is set to display only the
frontpage
        frontscreen();         //code block(function) written to display the content of front page
    }
    else {                  //'fs' is set to zero when 't' key is pressed on the keyboard
        target();          //The code block(function) where the targets(blocks) fro every level is
present in
        platform(pp);      //the code block(function) where the code for 12 positions of
platform is written in pp->platform position
        winF();              //code to check if any level has been won
        loss();              //code to check if any level has been lost
        alwaysDisplay();       //alwaysDisplay which display total moves & border of the
game
        glFlush();
    }
}
```

```
void init()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glClearColor(1, 0.9, 0.9, 0);
   glLoadIdentity();
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(0, 480, 0, 1000);
}

void keys(unsigned char k, int x, int y){
   switch (k) {
   case 'n': /* new game */

      lev = 1;
      //set the win or loss counter to zero
      win2 = 0;

//all win needs to be set to zero or the remain in same position even when new game is
clicked
      win = 0;        //same
      win3 = 0;    //same
      pp = 6;     //pp -> platform position should be set to the middle of the screen which is
6th position/12 positions
      pm2 = 0;     //platform movement counter should be reset
      pm1 = 0;     //platform movement counter should be reset
      pm3 = 0;     //platform movement counter should be reset

      ww1 = 0; //make the w key usable
      //The level 2 targets are set to '1' which means they are all present
      count1 = 1, count2 = 1, count3 = 1, count4 = 1, count5 = 1, count6 = 1, count7 = 1,
count8 = 1;
```

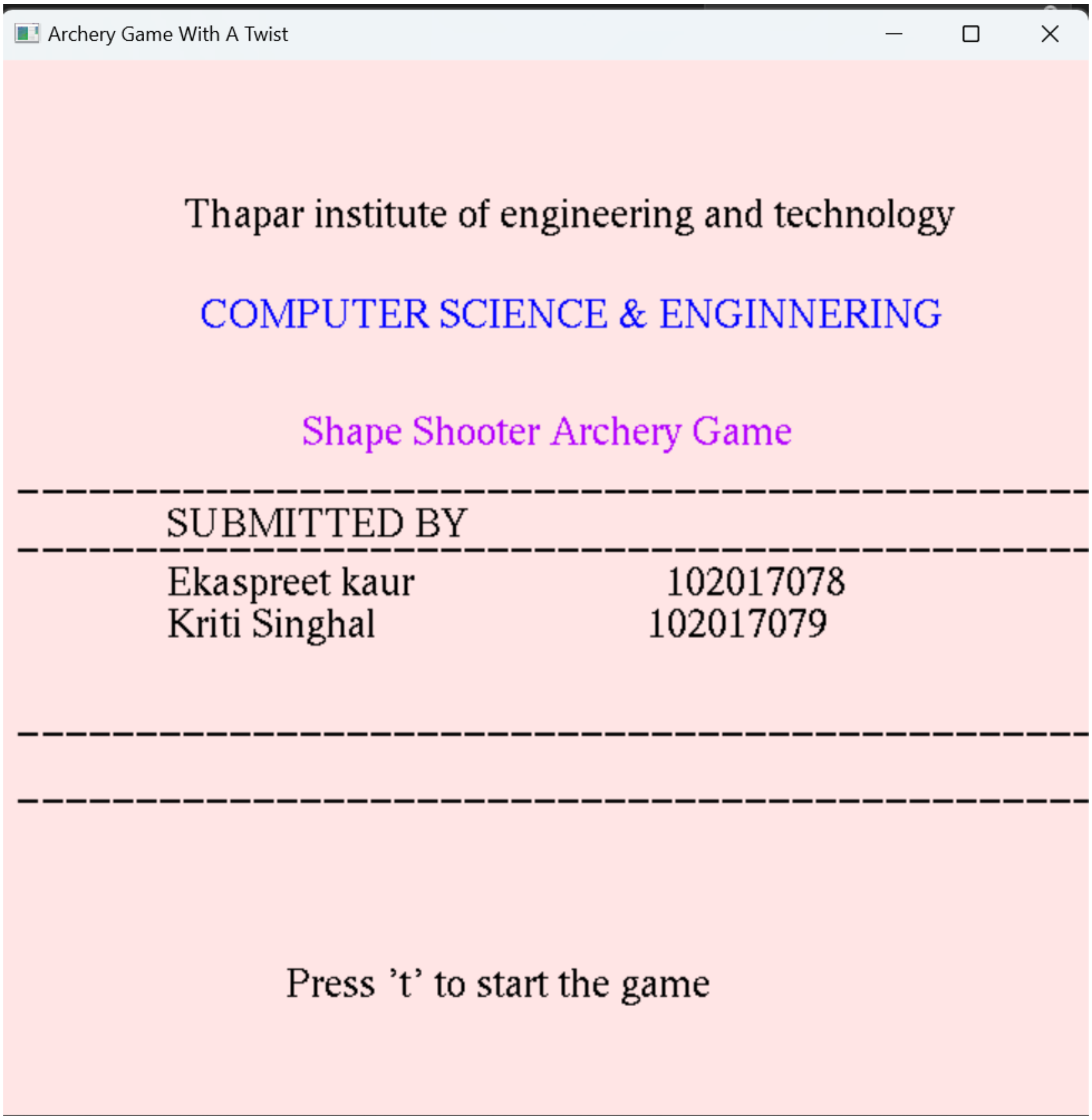
```
        break;
    case 'a':
        if (lev == 1) {     //In level one, increase the count(movement of blocks) pm->Platform 
movemnt
            pm1++;
        }
        if (lev == 2) {     //In level two, increase the count(movement of blocks) pm->Platform 
movemnt
            pm2++;
        }
        if (lev == 3) {     //In level three, increase the count(movement of blocks) pm-
>Platform movemnt
            pm3++;
        }
        pp = pp - 1;
        glutPostRedisplay();
        break;
    case 'd':
        if (lev == 1) {
            pm1++;
        }
        if (lev == 2) {
            pm2++;
        }
        if (lev == 3) {
            pm3++;
        }
        pp = pp + 1;
        glutPostRedisplay();
        break;
    case 'm':
        gs = 1;    //the m-> new game should disappear
        lev = 2;
        win2 = 0;  //all win needs to be set to zero or the remain in same position even when
new game is clicked
        win = 0;   //same
        win3 = 0;  //same
        pp = 6;
        pm2 = 0;
        pm1 = 0;
        pm3 = 0;
        count21 = 1, count22 = 1, count23 = 1, count24 = 1, count25 = 1;
        ww2 = 0;
        glutPostRedisplay();
        break;


    case 'b':
        lev = 3;
        win2 = 0;  //all win needs to be set to zero or the remain in same position even when
new game is clicked
        win = 0;          //same
        win3 = 0;     //same
        pp = 6;
        pm2 = 0;
        pm1 = 0;
        pm3 = 0;
        ww3 = 0;
        count31 = 1, count32 = 1, count33 = 1, count34 = 1, count35 = 1, count36 = 1;
        break;
    case 't':
        fs = 0;
    case 'w': //fire arrow (destroy targets)
        //pp->platform position and countXX ->block present('1') or not('0')
        //level 1 destroy blocks
```

```
if (ww1 == 0) {
    if (pp == 3 && count1 == 1) {
        count1 = 0;
        win += 1;
    }
    if (pp == 3 && count2 == 1) {
        count2 = 0;
        win += 1;
    }
    if (pp == 4 && count3 == 1) {
        count3 = 0;
        win += 1;
    }
    if (pp == 6 && count4 == 1) {
        count4 = 0;
        win += 1;
    }
    if (pp == 7 && count5 == 1) {
        count5 = 0;
        win += 1;
    }
    if (pp == 8 && count6 == 1) {
        count6 = 0;
        win += 1;
    }
    if (pp == 9 && count7 == 1) {
        count7 = 0;
        win += 1;
    }
    if (pp == 10 && count8 == 1) {
        count8 = 0;
        win += 1;
    }
    glutPostRedisplay();
}
//level 2 destroy blocks
if (ww2 == 0) {
    if (pp == 4 && count21 == 1) {
        count21 = 0;
        win2 += 1;
    }
    if (pp == 2 && count22 == 1) {
        count22 = 0;
        win2 += 1;
    }
    if (pp == 6 && count23 == 1) {
        count23 = 0;
        win2 += 1;
    }
    if (pp == 7 && count25 == 1) {
        count25 = 0;
        win2 += 1;
    }
    if (pp == 9 && count24 == 1) {
        count24 = 0;
        win2 += 1;
    }
    glutPostRedisplay();
}
```

```
//level 3 destroy blocks

if (ww3 == 0) {
    if (pp == 3 && count31 == 1) {
        count31 = 0;
        win3 += 1;
    }
    if (pp == 4 && count32 == 1) {
        count32 = 0;
        win3 += 1;
    }
    if (pp == 9 && count33 == 1) {
        count33 = 0;
        win3 += 1;
    }
    if (pp == 7 && count34 == 1) {
        count34 = 0;
        win3 += 1;
    }
    if (pp == 8 && count35 == 1) {
        count35 = 0;
        win3 += 1;
    }
    if (pp == 10 && count36 == 1) {
        count36 = 0;
        win3 += 1;
    }
    glutPostRedisplay();
}

break;
```

```
    case 'q':                    /* quit game */
        exit(0);
    }
}

int main(int argc, char** argv)      //main function of the game
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(650, 640);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Archery Game With A Twist");
    init();
    glutDisplayFunc(display);
    glutKeyboardFunc(keys);
    glutReshapeFunc(reshape);
    glutMainLoop();
    return 0;
}
```

SCREENSHOTS :



Archery Game With A Twist

Thapar institute of engineering and technology

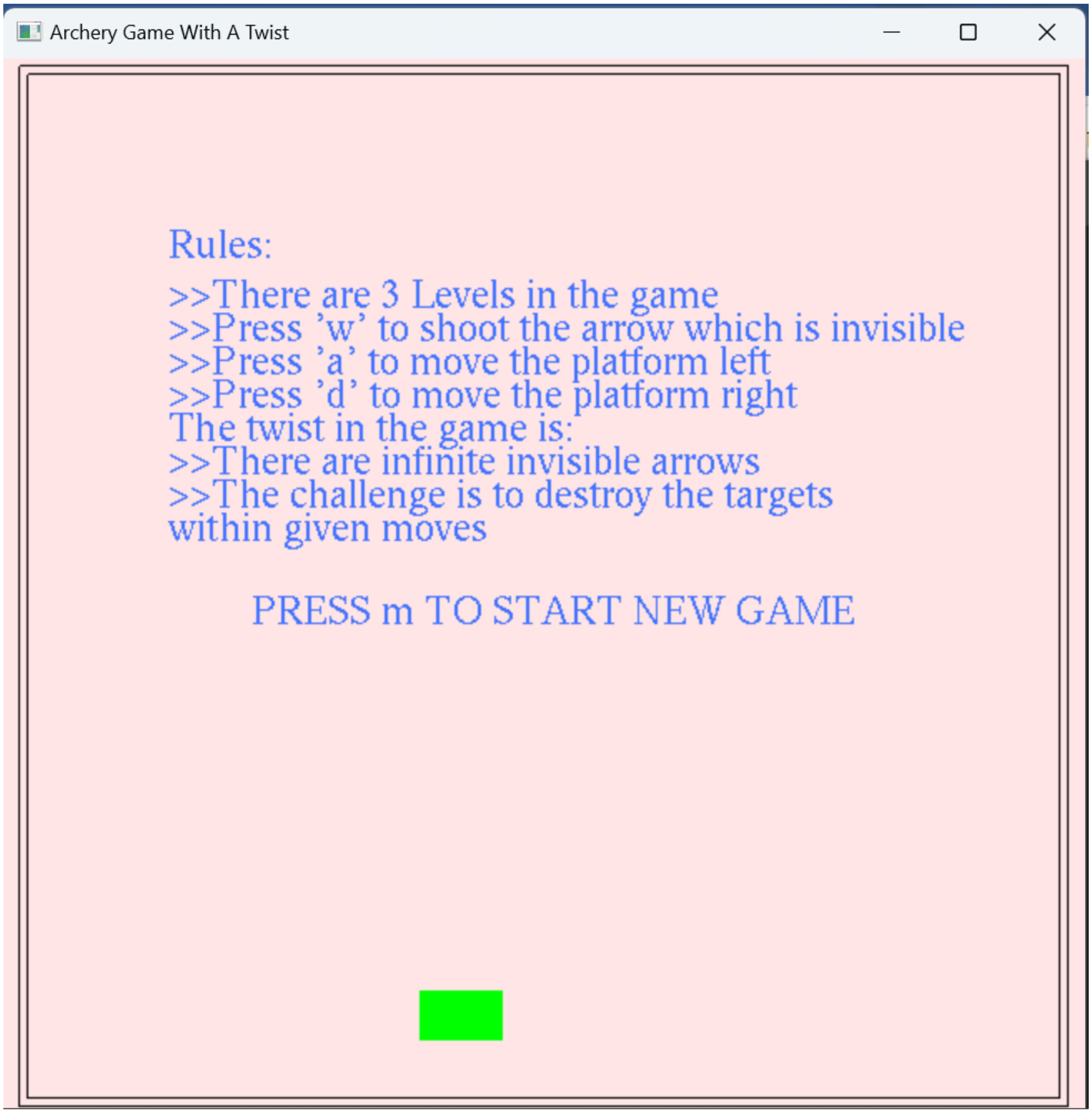
COMPUTER SCIENCE & ENGINNERING

Shape Shooter Archery Game

SUBMITTED BY

Ekaspreet kaur	102017078
Kriti Singhal	102017079

Press ’t’ to start the game



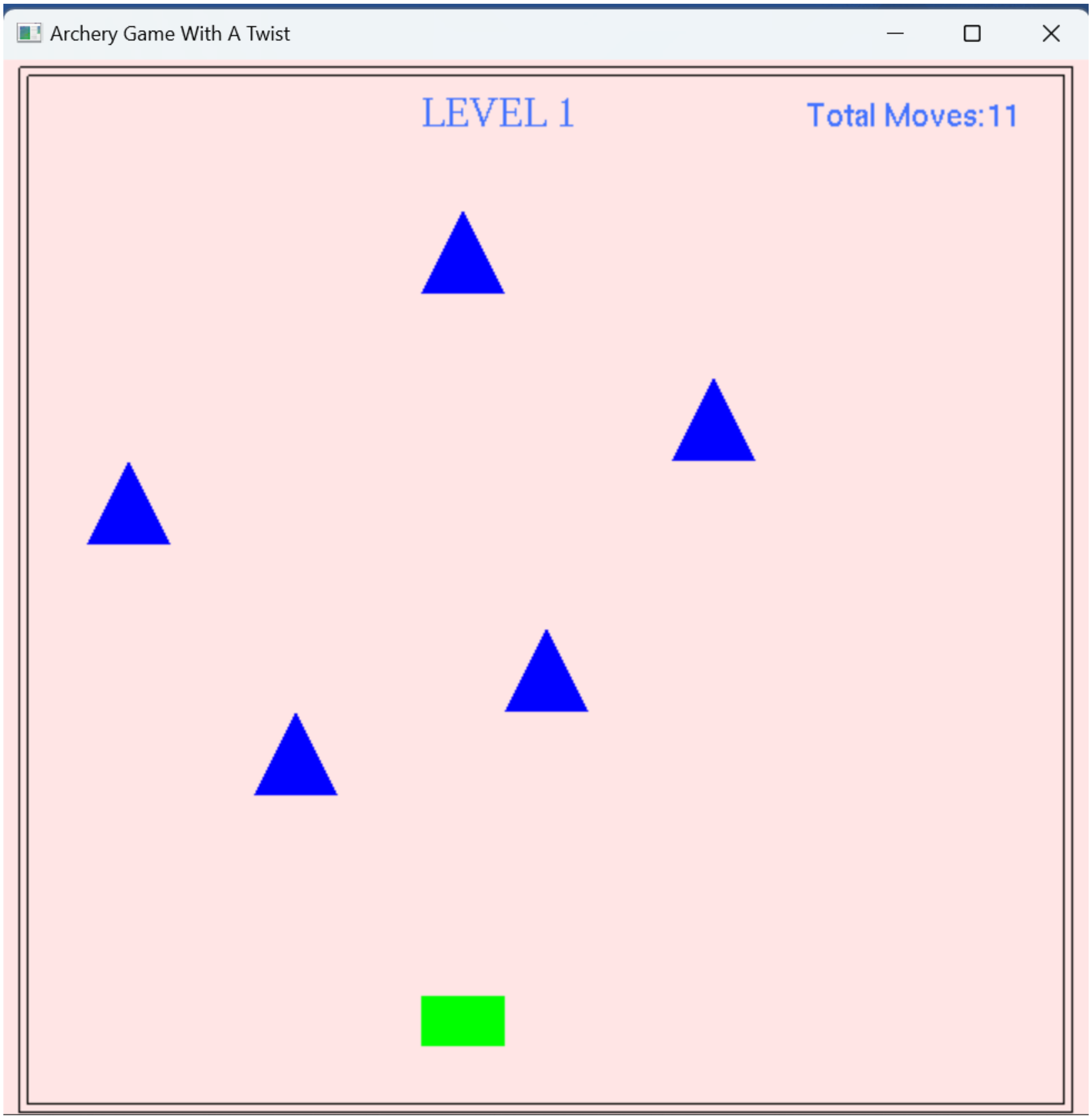
Archery Game With A Twist

Rules:

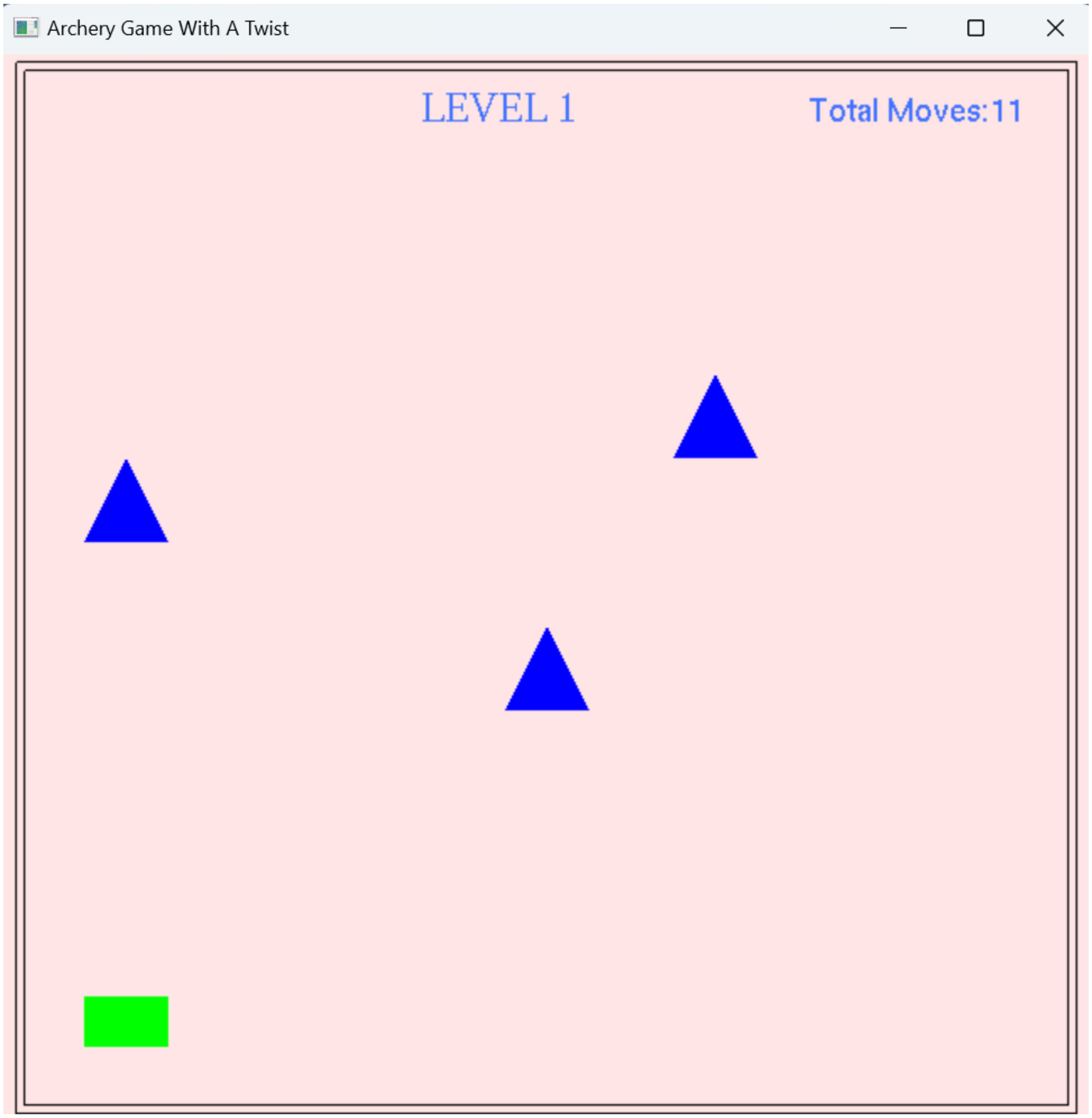
>>There are 3 Levels in the game
>>Press ’w’ to shoot the arrow which is invisible
>>Press ’a’ to move the platform left
>>Press ’d’ to move the platform right
The twist in the game is:
>>There are infinite invisible arrows
>>The challenge is to destroy the targets
within given moves

PRESS m TO START NEW GAME

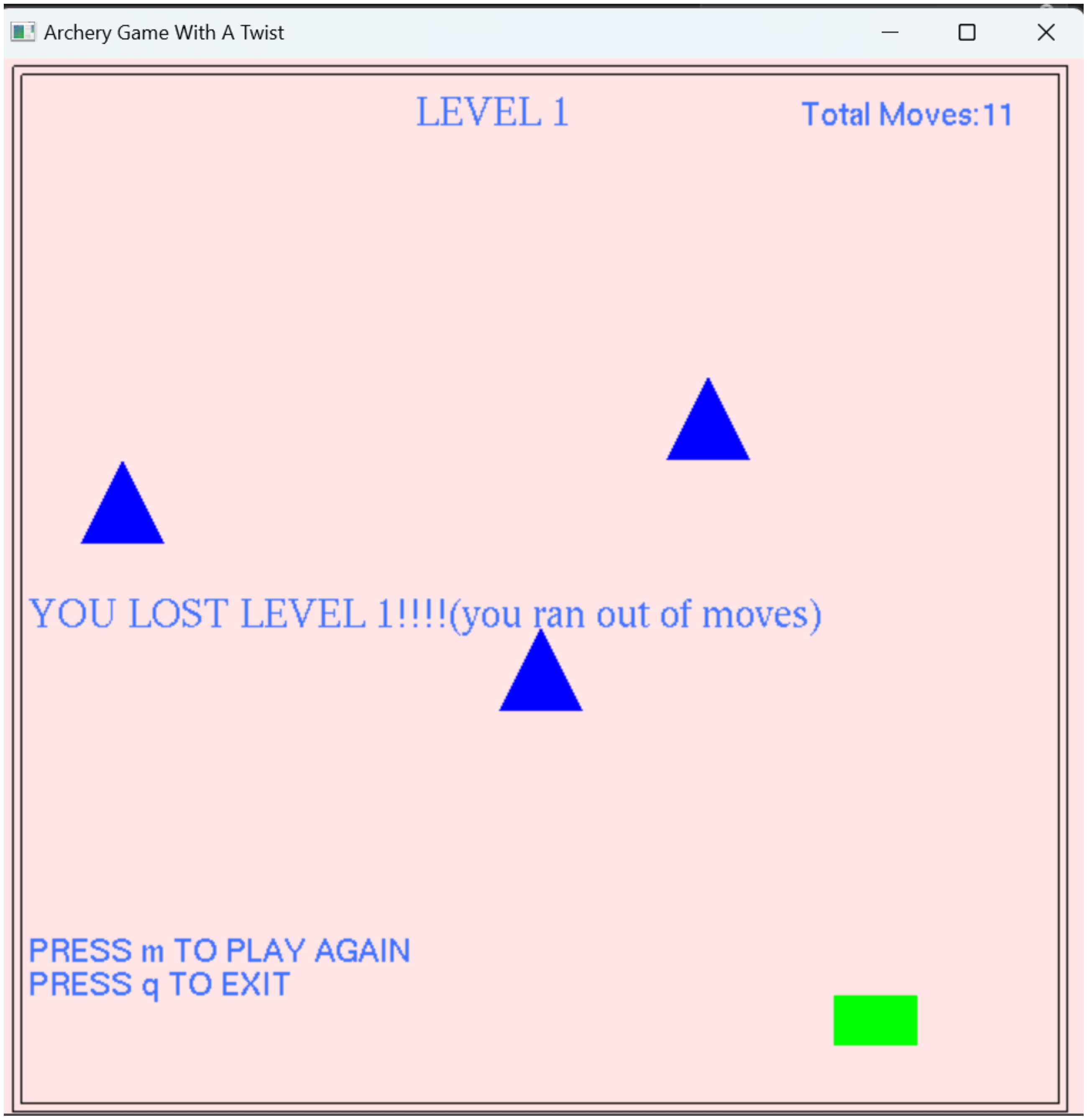
LEVEL 1:



While playing level 1



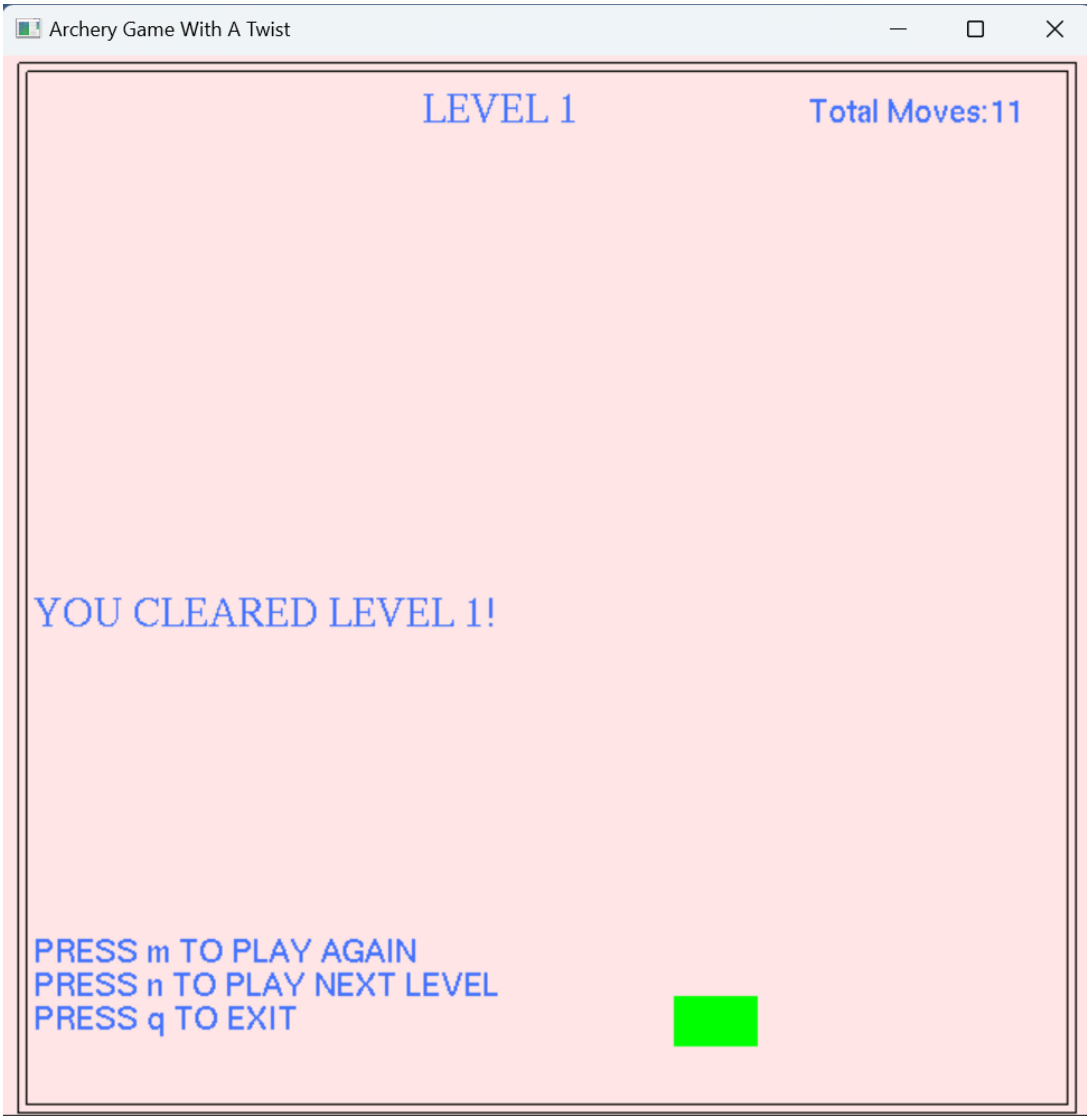
If player exceeds the total moves :



YOU LOST LEVEL 1!!!!(you ran out of moves)

PRESS m TO PLAY AGAIN
PRESS q TO EXIT

On shooting all shapes within the given moves, You clear the level 1.



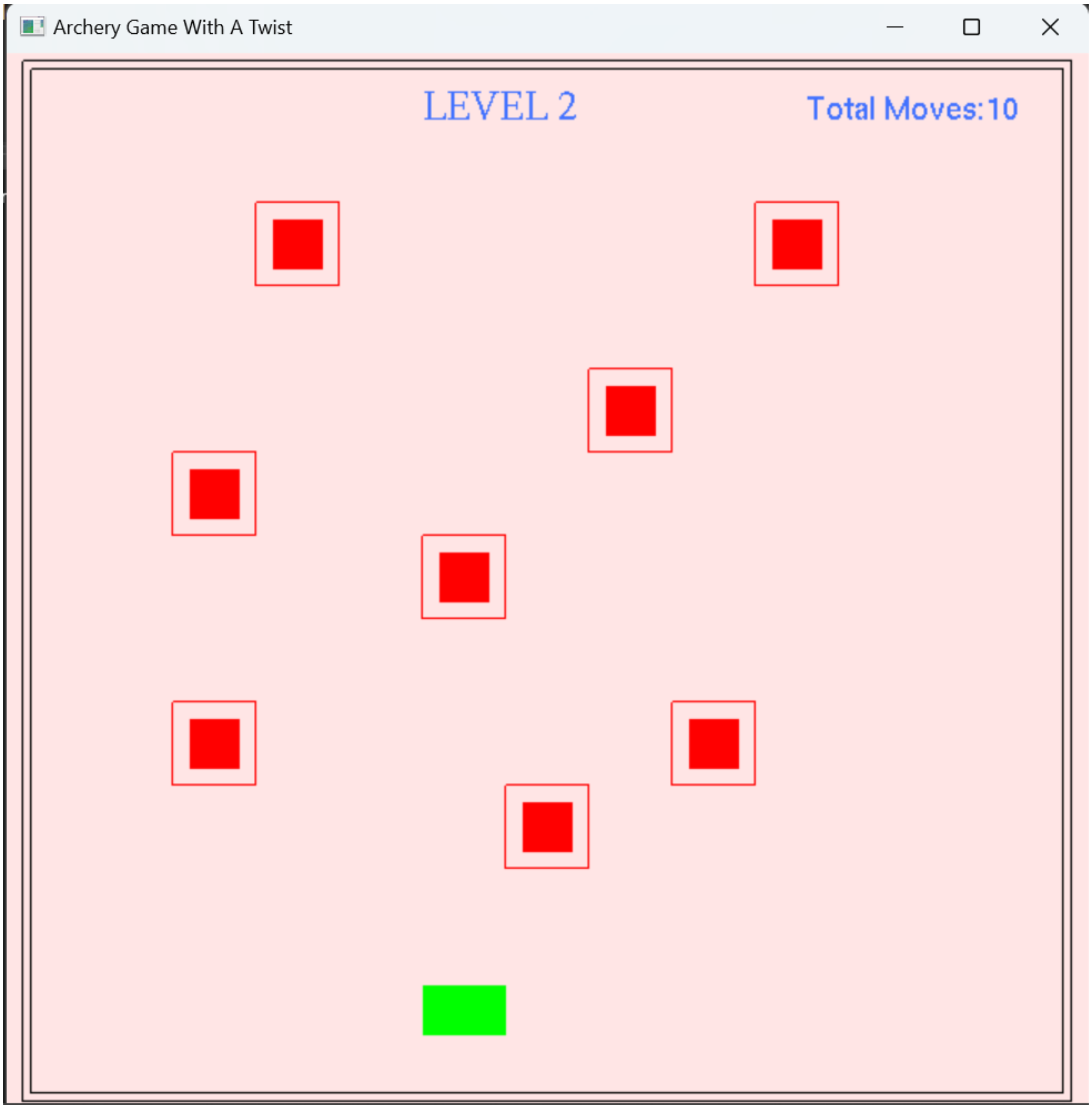
LEVEL 1

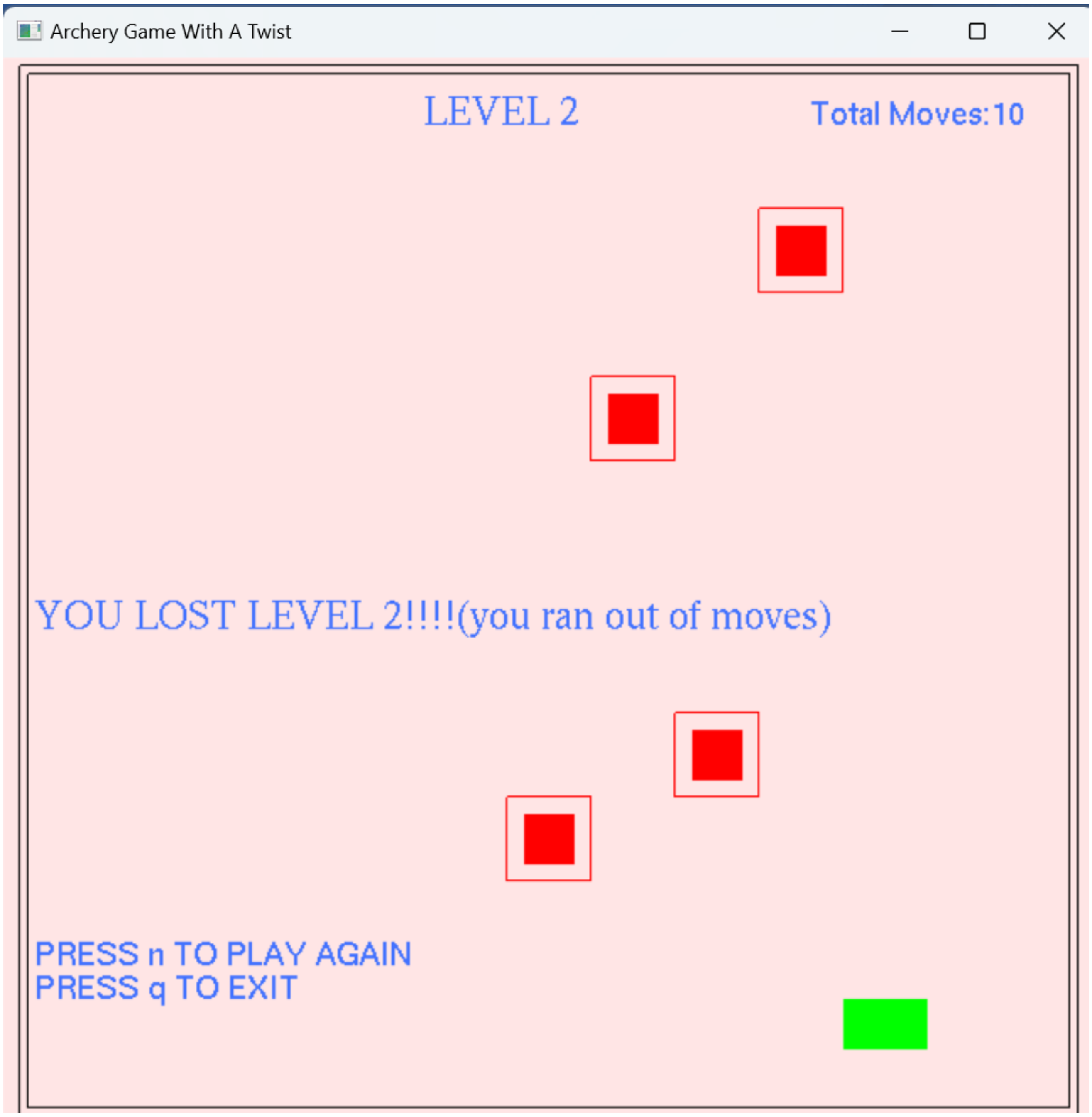
Total Moves:11

YOU CLEARED LEVEL 1!

PRESS m TO PLAY AGAIN
PRESS n TO PLAY NEXT LEVEL
PRESS q TO EXIT

LEVEL 2:



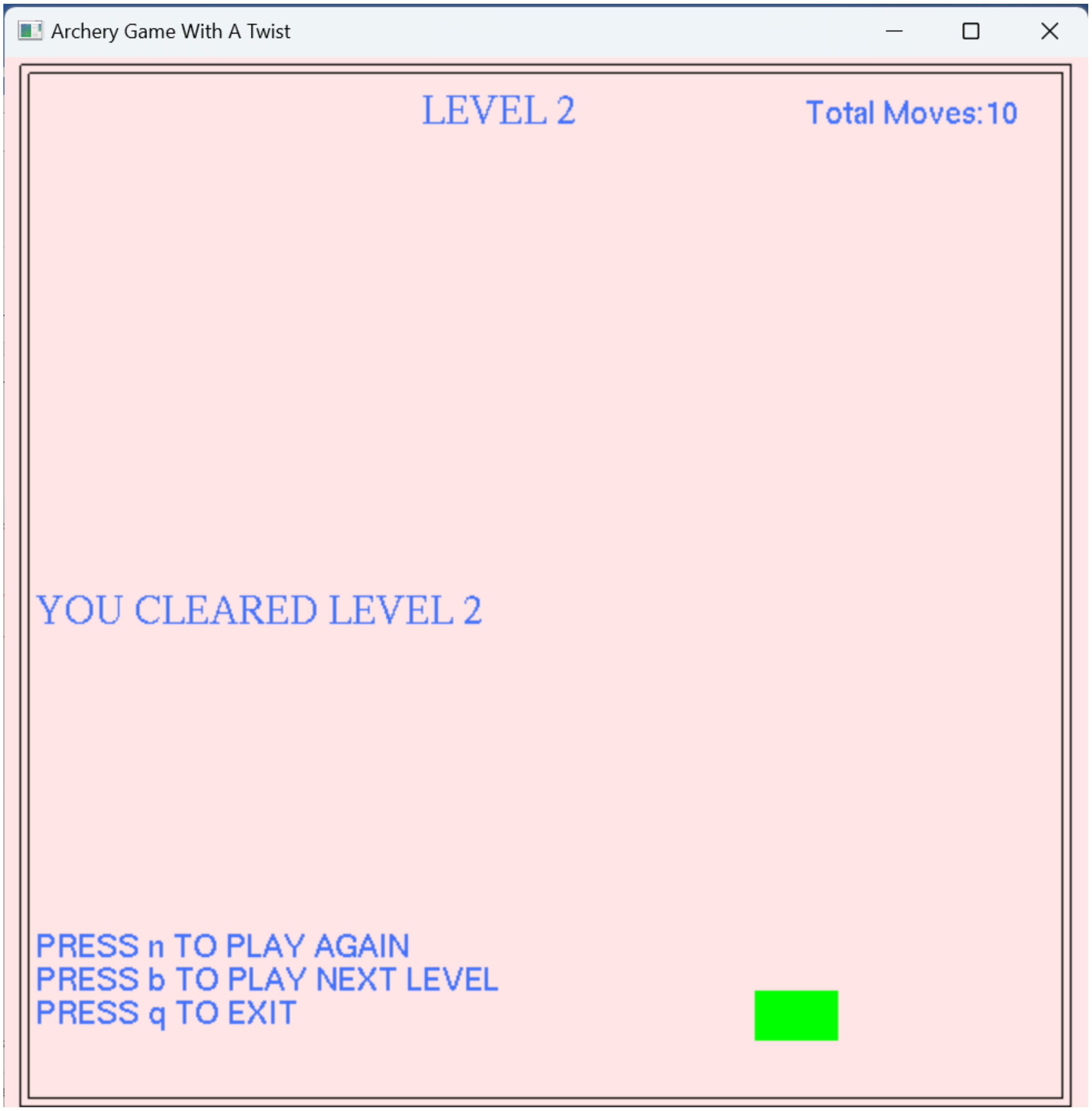

Archery Game With A Twist

LEVEL 2

Total Moves:10

YOU LOST LEVEL 2!!!!(you ran out of moves)

PRESS n TO PLAY AGAIN
PRESS q TO EXIT



Archery Game With A Twist

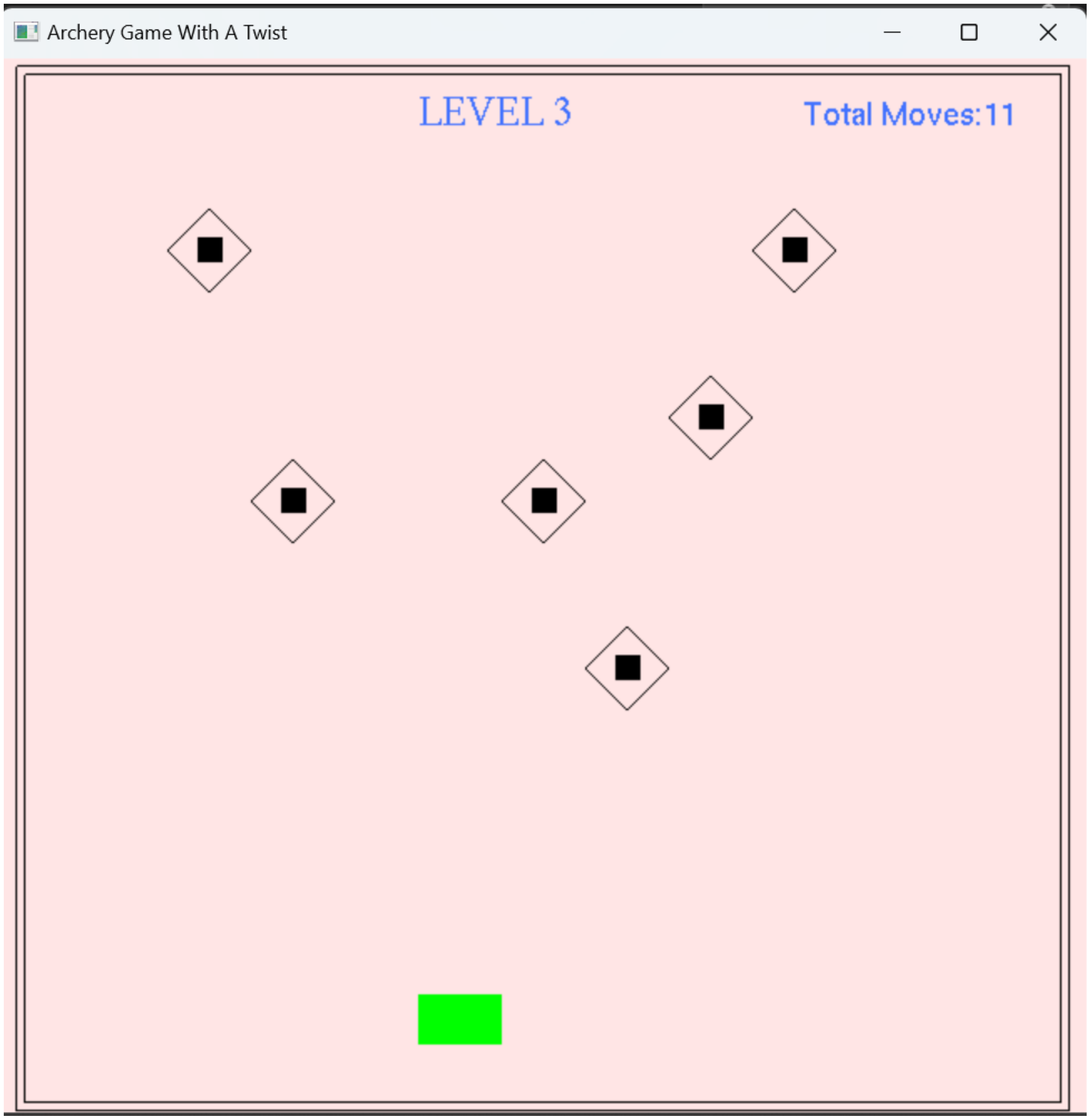
LEVEL 2

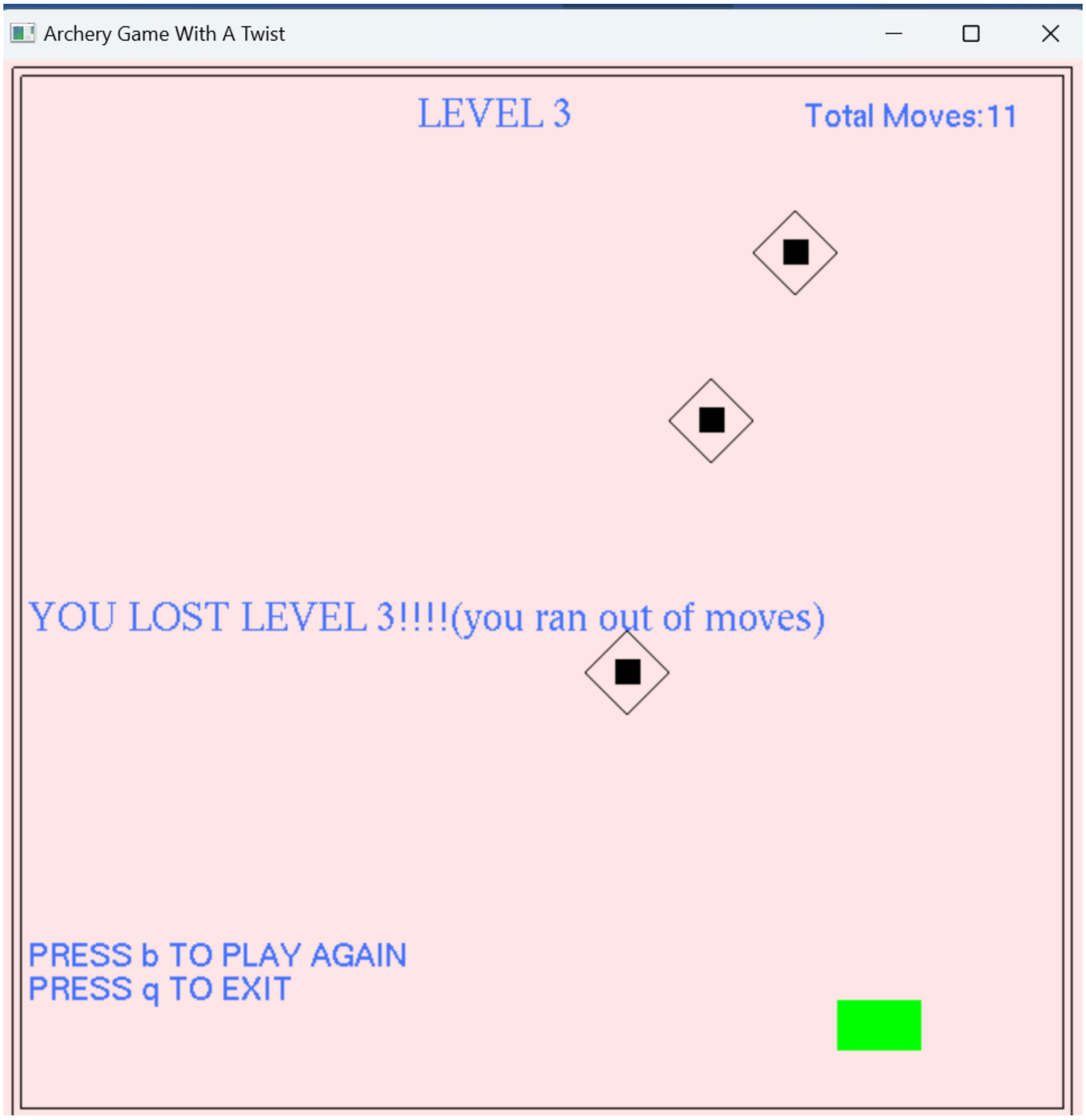
Total Moves:10

YOU CLEARED LEVEL 2

PRESS n TO PLAY AGAIN
PRESS b TO PLAY NEXT LEVEL
PRESS q TO EXIT

LEVEL 3:





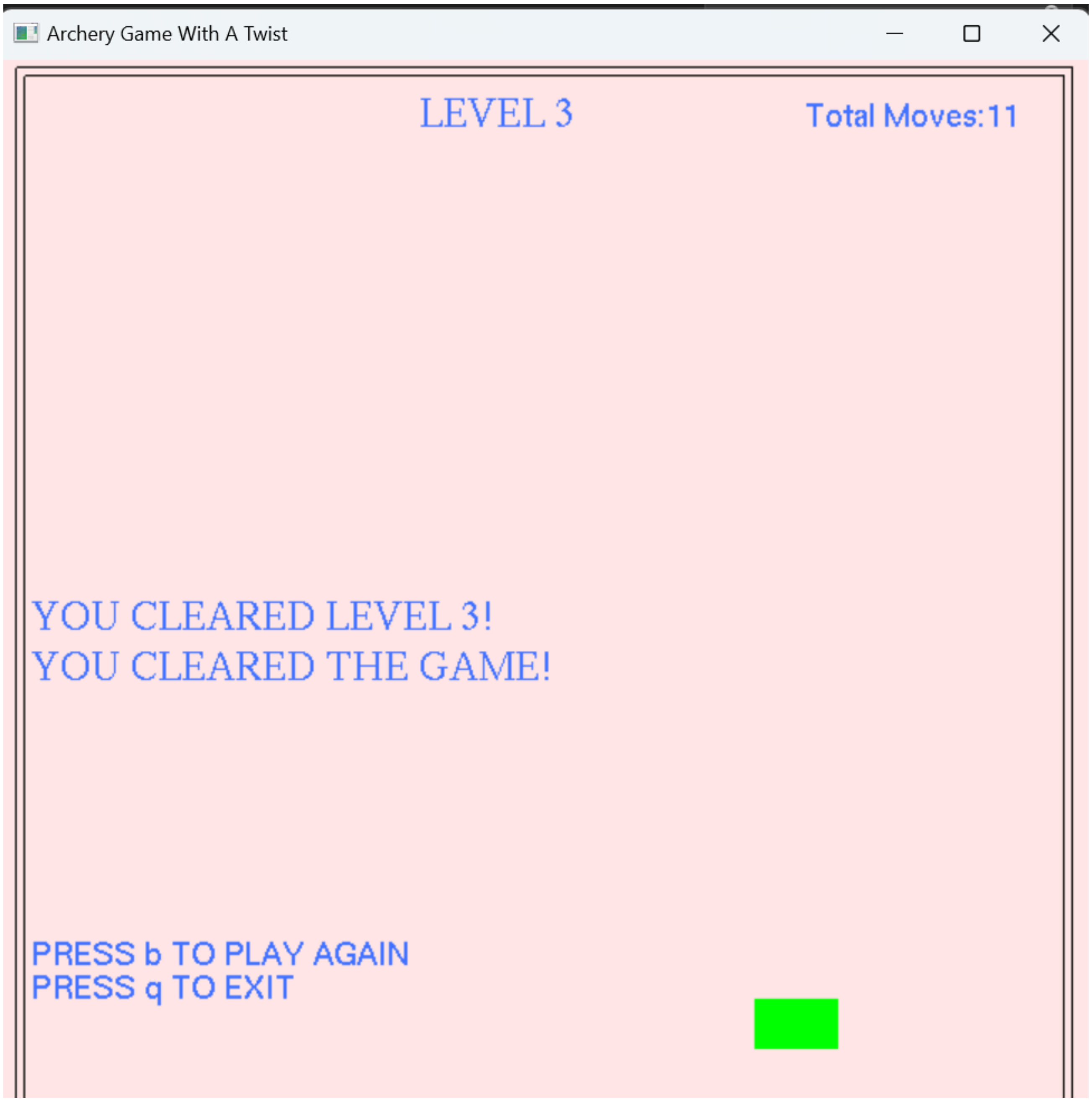
Archery Game With A Twist

LEVEL 3

Total Moves:11

YOU LOST LEVEL 3!!!!(you ran out of moves)

PRESS b TO PLAY AGAIN
PRESS q TO EXIT



LEVEL 3

Total Moves:11

YOU CLEARED LEVEL 3!
YOU CLEARED THE GAME!

PRESS b TO PLAY AGAIN
PRESS q TO EXIT