```cpp
/*
 * NAME: Jiayi Wang
 *
 * Fall 2015 CS349 Assignment 1:  An implementation of Breakout in C/C++ and Xlib.
 *
 *
 *
 * Commands to compile and run:
 *
 *  g++ -o a1 a1.cpp -L/usr/X11R6/lib -lX11 -lstdc++
 *  ./a1
 *
 * Note: the -L option and -lstdc++ may not be needed on some machines.
 */


#include <iostream>
#include <list>
#include <cstdlib>
#include <sys/time.h>
#include <math.h>
#include <stdio.h>
#include <unistd.h>
#include <vector>
#include <sstream>


/*
 * Header files for X functions
 */
#include <X11/Xlib.h>
#include <X11/Xutil.h>

using namespace std;

/* information of drawing, cannot change brick size */
const int Border = 5;
const int Brick_width = 80;
const int Brick_height = 45;
const int Gap = 2;
const int BallDiameter = 10;
const int Paddle_height = 5;

/* location and speed */
int paddle_x = 0;
int paddle_x_previous = 0;
int paddle_y = 0;
int ball_x = 0;
int ball_y = 0;
int Paddle_length = 100;

/* Global variables */
int FPS = 30;   // frame per second
int Speed = 5;  // pixel per second
int Score = 0;
int score_per_brick = 10;
int Window_width = 800;  // init width
int Window_height = 600;  // init height
const int BufferSize = 10;
bool gameOver = false;
bool gameWon = false;
int total_rows = 0; // rows of bricks
int total_cols = 0; // cols of bricks


/* Power-Up  types */
int current_power_up = 0;
const int PU_LONG_PADDLE = 1;
const int PU_SHORT_PADDLE = 2;
const int PU_SLOW_BALL = 3;
const int PU_FAST_BALL = 4;
const int PU_DOUBLE_SCORE = 5;

/*
 * Information to draw on the window.
 */
struct XInfo {
    Display  *display;
    int      screen;
    Window   window;
    GC       gc[9];

    Pixmap  pixmap;     // double buffer
    int     width;      // size of pixmap
    int     height;
};

class BrickInfo {
public:
    BrickInfo(int row, int col): row(row), col(col) {
        int random_result = rand() % 30;
        power_up_type = (random_result < 6) ? random_result : 0;
        if_exist = true;
    }

    int     row;
    int     col;
    int     power_up_type;
    bool    if_exist;
};
std::vector<BrickInfo> BrickInfoVector;


/*
 * Function to put out a message on error exits.
 */
void error( string str ) {
  cerr << str << endl;
  exit(0);
}



/*
 * An abstract class representing displayable things.
 */
class Displayable {
    public:
        virtual void paint(XInfo &xinfo) = 0;
};


class Bricks : public Displayable {
    public:
        virtual void paint(XInfo &xinfo) {
            for (std::vector<BrickInfo>::iterator it = BrickInfoVector.begin() ; it != BrickInfoVector.end(); ++it){
                BrickInfo bi = *it;
                if (! bi.if_exist) continue;

                if (bi.power_up_type != 0) {//power up
                    XFillRectangle( xinfo.display, xinfo.pixmap, xinfo.gc[2],
                        bi.col * (Brick_width + Gap), //x
                        bi.row * (Brick_height + Gap), //y
                        Brick_width, Brick_height );
                } else {
                    int colour_type = (bi.row % 3) + 6;
                    XFillRectangle( xinfo.display, xinfo.pixmap, xinfo.gc[colour_type],
                        bi.col * (Brick_width + Gap), //x
                        bi.row * (Brick_height + Gap), //y
                        Brick_width, Brick_height );
                }
            }

        }
        // constructor
        Bricks(int null) {
            BrickInfoVector.clear();

            total_rows = 6;

            total_cols = (Window_width / (Brick_width + Gap)) + 1;

            for (int row = 0; row < total_rows; row++) {
                for (int col = 0; col < total_cols; col++) {
                    BrickInfo bi = BrickInfo(row,col);
                    BrickInfoVector.push_back(bi);
                }
            }
        }

};

class Ball : public Displayable {
    public:
        virtual void paint(XInfo &xinfo) {
            XFillArc(xinfo.display, xinfo.pixmap, xinfo.gc[3], x, y, BallDiameter, BallDiameter, 0, 360*64);
    }

    void move(XInfo &xinfo) {

        // hit brick
        if ( !hit_brick ) {
            int temp_col = x / (Brick_width + Gap);
            int temp_row = y / (Brick_height + Gap);
            BrickInfo bi = BrickInfoVector[temp_row * total_cols + temp_col];

            if ( bi.if_exist ) {
                if (bi.power_up_type != 0) {
                    current_power_up = bi.power_up_type;
                    switch (current_power_up) {
                        case 1: Paddle_length *= 2; break;
                        case 2: Paddle_length /= 2; break;
                        case 3: speed_x *= 2; speed_y *= 2; break;
                        case 4: speed_x /= 2; speed_y /= 2; break;
                        case 5: score_per_brick *= 2; break;
                        default: break;
                    }
                }

                speed_y = 0 - speed_y;
                Score += score_per_brick;
                bi.if_exist = false;
                hit_brick = true;
            } // if ( bi.if_exist )

        } else {
            hit_brick = false;
        }

        // hit paddle
        if ( (!hit_paddle) && ( y + BallDiameter >= paddle_y) &&
             (y <= paddle_y+Paddle_height) && (x+BallDiameter >= paddle_x) &&
             (x <= paddle_x+Paddle_height) ) {
            hit_paddle = true;
            speed_y = 0 - speed_y;

            speed_x = speed_x + (paddle_x - paddle_x_previous);
            if (speed_x > 0 && (speed_x > 1.5 * Speed)) speed_x = 1.5 * Speed;
            if (speed_x < 0 && (speed_x < 1.5 * Speed)) speed_x = 1.5 * Speed;
        } else {
            hit_paddle = false;
        }

        // hit wall
        if ( !hit_wall ) {

            if (x <= 0) {
                speed_x = 0 - speed_x;
                x = 0;
                hit_wall = true;
            } else if ( x + BallDiameter >= Window_width) {
                speed_x = 0 - speed_x;
                x = Window_width - BallDiameter;
                hit_wall = true;
            }

            if (y <= 0) {
                speed_y = 0 - speed_y;
                y = 0;
                hit_wall = true;
            } else if ( y + BallDiameter >= Window_height) {
                hit_wall = true;
                gameOver = true;
                return;
            }
        } else {
            hit_wall = false;
        }

        x = x + speed_x;
        y = y + speed_y;
    }

    Ball(int null) {
        x = (Window_width - BallDiameter)/2;
        y = (Window_height/1.2)-BallDiameter;
        speed_x = 0;
        speed_y = Speed;
        hit_wall = false;
```

```
            hit_paddle = false;
            hit_brick = false;
        }

    private:
        int x;
        int y;
        int speed_x;
        int speed_y;
        bool hit_wall;
        bool hit_paddle;
        bool hit_brick;
};


class Paddle : public Displayable {
   public:
      virtual void paint(XInfo &xinfo ) {
        XFillRectangle( xinfo.display, xinfo.pixmap, xinfo.gc[1], paddle_x, paddle_y, Paddle_length, Paddle_height );
      }

      void moveLeft( XInfo &xinfo ) {
        paddle_x_previous = paddle_x;
        paddle_x = ( paddle_x - Speed < 0) ? 0 : paddle_x - Speed;
      }

      void moveRight( XInfo &xinfo ) {
        paddle_x_previous = paddle_x;
        paddle_x = ( paddle_x + Paddle_length + Speed > Window_width ) ?
                    Window_width - Paddle_length : paddle_x + Speed - Paddle_length;
      }

      void moveAnywhere ( int x ) {
        paddle_x_previous = paddle_x;

         if ( x <= Paddle_length/2 ) {
            paddle_x = 0;
         } else if ( x >= Window_width - (Paddle_length/2) ) {
            paddle_x = Window_width - Paddle_length;
         } else {
            paddle_x = x - (Paddle_length/2);
         }

      }
    // constructor
    Paddle (int null) {
        paddle_x = (Window_width - Paddle_length) /2;
        paddle_y = Window_height/1.2;
        paddle_x_previous = paddle_x;
    }
};

list<Displayable *> dList;           // list of Displayables
Bricks bricks(0);
Ball ball(0);
Paddle paddle(0);

/*
 * Create a window
 */
void initX(int argc, char* argv[], XInfo& xInfo) {

    XSizeHints hints;
    hints.x = 100;
    hints.y = 100;
    hints.width = 10 * Brick_width + 9 * Gap;
    hints.height = 6 * Brick_height + 5 * Gap + 100;
    hints.min_width = 5 * Brick_width + 4 * Gap;
    hints.min_height = 5 * Brick_height + 4 * Gap + 100;
    hints.flags = PPosition | PSize;


    /*
    * Display opening uses the DISPLAY    environment variable.
    * It can go wrong if DISPLAY isn't set, or you don't have permission.
    */
    xInfo.display = XOpenDisplay( "" );
    if ( !xInfo.display )    {
        error( "Can't open display." );
    }

    /*
    * Find out some things about the display you're using.
    */
    xInfo.screen = DefaultScreen( xInfo.display ); // macro to get default screen index

    unsigned long white, black;
    white = XWhitePixel( xInfo.display, xInfo.screen );
    black = XBlackPixel( xInfo.display, xInfo.screen );

    xInfo.window = XCreateSimpleWindow(
        xInfo.display,                // display where window appears
        DefaultRootWindow( xInfo.display ), // window's parent in window tree
        hints.x, hints.y,               // upper left corner location
        hints.width, hints.height,   // size of the window
        Border,                             // width of window's border
        black,                        // window border colour
        white );                        // window background colour

    // extra window properties like a window title
    XSetStandardProperties(
        xInfo.display,        // display containing the window
        xInfo.window,        // window whose properties are set
        "Breakout",    // window's title
        "OW",                // icon's title
        None,                // pixmap for the icon
        argv, argc,            // applications command line args
        &hints );            // size hints for the window

    /* Colormap */
    Colormap screen_colormap;
    XColor red, blue, yellow, green, grey;
    screen_colormap = DefaultColormap(xInfo.display, xInfo.screen);
    XAllocNamedColor(xInfo.display, screen_colormap, "red", &red, &red);
    XAllocNamedColor(xInfo.display, screen_colormap, "blue", &blue, &blue);
    XAllocNamedColor(xInfo.display, screen_colormap, "yellow", &yellow, &yellow);
    XAllocNamedColor(xInfo.display, screen_colormap, "green", &green, &green);
    XAllocNamedColor(xInfo.display, screen_colormap, "grey", &grey, &grey);

    /*
     * Create Graphics Contexts
     */
    int i = 0;
    xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
    XSetForeground( xInfo.display, xInfo.gc[i], white );
    XSetBackground( xInfo.display, xInfo.gc[i], black );
    XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid );
    XSetLineAttributes( xInfo.display, xInfo.gc[i],
                        1, LineSolid, CapButt, JoinRound );

    // Reverse Video
    i = 1;
    xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
    XSetForeground( xInfo.display, xInfo.gc[i], red.pixel );
    XSetBackground( xInfo.display, xInfo.gc[i], white );
    XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid);
    XSetLineAttributes( xInfo.display, xInfo.gc[i],
                         1, LineOnOffDash, CapRound, JoinRound );

    i = 2; // power up (as a red brick)
    xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
    XSetForeground( xInfo.display, xInfo.gc[i], red.pixel );
    XSetBackground( xInfo.display, xInfo.gc[i], white );
    XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid );
    XSetLineAttributes( xInfo.display, xInfo.gc[i], Gap, LineSolid, CapRound, JoinMiter );

    i = 3; // grey ball
    xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
    XSetForeground( xInfo.display, xInfo.gc[i], red.pixel );
    XSetBackground( xInfo.display, xInfo.gc[i], white );
    XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid );
    XSetLineAttributes( xInfo.display, xInfo.gc[i], 1, LineSolid, CapRound, JoinRound );

    i = 4; // black text
    xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
    XSetForeground( xInfo.display, xInfo.gc[i], black );

    i = 5; // red text
    xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
    XSetForeground( xInfo.display, xInfo.gc[i], red.pixel );

    i = 6; // blue brick
    xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
    XSetForeground( xInfo.display, xInfo.gc[i], blue.pixel );
    XSetBackground( xInfo.display, xInfo.gc[i], white );
    XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid );
    XSetLineAttributes( xInfo.display, xInfo.gc[i], Gap, LineSolid, CapRound, JoinMiter );

    i = 7; // yellow brick
    xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
    XSetForeground( xInfo.display, xInfo.gc[i], yellow.pixel );
    XSetBackground( xInfo.display, xInfo.gc[i], white );
    XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid );
    XSetLineAttributes( xInfo.display, xInfo.gc[i], Gap, LineSolid, CapRound, JoinMiter );

    i = 8; // green brick
    xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
    XSetForeground( xInfo.display, xInfo.gc[i], green.pixel );
    XSetBackground( xInfo.display, xInfo.gc[i], white );
    XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid );
    XSetLineAttributes( xInfo.display, xInfo.gc[i], Gap, LineSolid, CapRound, JoinMiter );

    /* Font */
    XFontStruct* font_info1;
    XFontStruct* font_info2;
    string font_name1 = "-*-*-*-*-*--14-*-*";
    string font_name2 = "-*-*-*-*-*--20-*-*";
    font_info1 = XLoadQueryFont(xInfo.display, font_name1.c_str());
    font_info2 = XLoadQueryFont(xInfo.display, font_name2.c_str());
    XSetFont(xInfo.display, xInfo.gc[4], font_info1->fid);
    XSetFont(xInfo.display, xInfo.gc[5], font_info2->fid);

    /* pixmap */
    int depth = DefaultDepth(xInfo.display, DefaultScreen(xInfo.display));
    xInfo.pixmap = XCreatePixmap(xInfo.display, xInfo.window, hints.width, hints.height, depth);
    xInfo.width = hints.width;
    xInfo.height = hints.height;


    XSelectInput(xInfo.display, xInfo.window,
        KeyPressMask | PointerMotionMask | StructureNotifyMask );

    /*
     * Don't paint the background -- reduce flickering
     */
    XSetWindowBackgroundPixmap(xInfo.display, xInfo.window, None);

    /*
     * Put the window on the screen.
     */
    XMapRaised( xInfo.display, xInfo.window );

    XFlush(xInfo.display);
}

/*
 * Function to repaint a display list
 */
void repaint( XInfo &xinfo) {
    list<Displayable *>::const_iterator begin = dList.begin();
    list<Displayable *>::const_iterator end = dList.end();

    // draw into the buffer
    // note that a window and a pixmap are âdrawablesâ
    XFillRectangle(xinfo.display, xinfo.pixmap, xinfo.gc[0],
        0, 0, xinfo.width, xinfo.height);
    while( begin != end ) {
        Displayable *d = *begin;
        d->paint(xinfo); // the displayables know about the pixmap
        begin++;
    }
    // copy buffer to window
    XCopyArea(xinfo.display, xinfo.pixmap, xinfo.window, xinfo.gc[0],
        0, 0, xinfo.width, xinfo.height,  // region of pixmap to copy
        0, 0); // position to put top left corner of pixmap in window

    XFlush( xinfo.display );

}

void handleKeyPress(XInfo &xinfo, XEvent &event) {
    KeySym key;
    char text[BufferSize];

    int i = XLookupString(
        (XKeyEvent *)&event,    // the keyboard event
        text,                   // buffer when text will be written
        BufferSize,             // size of the text buffer
        &key,                   // workstation-independent key symbol
```

```cpp
        NULL );                 // pointer to a composeStatus structure (unused)
    if ( i == 1) {
        if (text[0] == 'q') {
            error("Terminating normally.");
            exit(0);
        } else if (text[0] == 'c') {
            gameWon = true;
            Score = 10000;
        } else if (text[0] == 'a') {
            paddle.moveLeft(xinfo);
        } else if (text[0] == 'd') {
            paddle.moveRight(xinfo);
        }
    }
}

void handleMotion(XInfo &xinfo, XEvent &event) {
    paddle.moveAnywhere(event.xbutton.x);
}

void handleResize(XInfo &xinfo, XEvent &event) {
    XConfigureEvent xce = event.xconfigure;
    fprintf(stderr, "Handling resize  w=%d  h=%d\n", xce.width, xce.height);
    if (xce.width > xinfo.width || xce.height > xinfo.height) {
        XFreePixmap(xinfo.display, xinfo.pixmap);
        int depth = DefaultDepth(xinfo.display, DefaultScreen(xinfo.display));
        xinfo.pixmap = XCreatePixmap(xinfo.display, xinfo.window, xce.width, xce.height, depth);
        xinfo.width = xce.width;
        xinfo.height = xce.height;
    }
}

void handleText(XInfo &xinfo) {

    stringstream score_ss;
    score_ss << Score;
    string score_string = score_ss.str();
    XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], 10, Window_height-30, "Score: ", 7 );
    XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], 60, Window_height-30, score_string.c_str(), score_string.length() );

    stringstream FPS_ss;
    FPS_ss << FPS;
    string FPS_string = FPS_ss.str();
    XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], Window_width-100, Window_height-30, "FPS: ", 5 );
    XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], Window_width-50, Window_height-30, FPS_string.c_str(), FPS_string.length() );

    string power_up_string;
    switch (current_power_up) {
        case 1: power_up_string = "Long paddle "; break;
        case 2: power_up_string = "Short paddle"; break;
        case 3: power_up_string = "Faster ball "; break;
        case 4: power_up_string = "Slower ball "; break;
        case 5: power_up_string = "Double score"; break;
        default: power_up_string = "None"; break;
    }
    XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], 10, Window_height-10, "Power up: ", 10 );
    XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], 80, Window_height-10, power_up_string.c_str(), power_up_string.length() );

    stringstream speed_ss;
    speed_ss << Speed;
    string speed_string = speed_ss.str();
    XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], Window_width-100, Window_height-10, "Ball speed: ", 12 );
    XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], Window_width-70, Window_height-10, speed_string.c_str(), speed_string.length() );

   if ( gameOver ) {
      XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[5], (Window_width/2)-45, Window_height/2, "GAME OVER", 9 );
   }

   if ( gameWon ) {
      XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[5], (Window_width/2)-45, Window_height/2, "Game WON", 8 );
   }

}

void splash_screen ( XInfo &xinfo ) {
    XFillRectangle(xinfo.display, xinfo.window, xinfo.gc[0], 0, 0, Window_width, Window_height);

    XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 50, 20, "Breakout Game", 13 );
    XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 50, "Name: Jiayi Wang", 16 );
    XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 70, "ID: 20501675", 12 );
    // blank line
    XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 140, "- Press 's' to start the game", 29 );
    XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 160, "- Press 'q' to quit the game", 28 );
    XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 180, "- Press 'c' to cheat(win immidiately)", 37 );
    // blank line
    XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 230, "You can either use your mouse,", 30 );
    XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 250, "or press A and D on keyboard to control the paddle.", 51 );
    // blank line
    XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 300, "Notice:", 7 );
    XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 320, "The red bricks are Power Ups you can get, but they can be debuff!", 65 );

    XFlush( xinfo.display );
}

int handleStart(XInfo &xinfo, XEvent &event) {
    KeySym key;
    char text[BufferSize];

    int i = XLookupString(
        (XKeyEvent *)&event,    // the keyboard event
        text,                   // buffer when text will be written
        BufferSize,             // size of the text buffer
        &key,                   // workstation-independent key symbol
        NULL );                 // pointer to a composeStatus structure (unused)
    if ( i == 1) {
        if (text[0] == 's') {
            return 1;
        }
    }
    return 0;
}

unsigned long now() {
    timeval tv;
    gettimeofday(&tv, NULL);
    return tv.tv_sec * 1000000 + tv.tv_usec;
}

void eventLoop(XInfo &xinfo) {
    // Add stuff to paint to the display list
    dList.push_front(&bricks);
    dList.push_front(&ball);
    dList.push_front(&paddle);

    XEvent event;
    unsigned long lastRepaint = 0;

    while(true) {
        int ret_val = 0;
        if (XPending(xinfo.display) > 0) {
            XNextEvent( xinfo.display, &event );
            if( event.type == KeyPress) ret_val = handleStart(xinfo, event);
        }
        if (ret_val == 1) break;
    }

    while( !gameOver && !gameWon ) {
        if (XPending(xinfo.display) > 0) {
            XNextEvent( xinfo.display, &event );
            switch( event.type ) {
                case KeyPress:
                    handleKeyPress(xinfo, event);
                    break;
                case MotionNotify:
                    handleMotion(xinfo, event);
                    break;
                case ConfigureNotify:
                    handleResize(xinfo, event);
                    break;
            }
        }

        usleep(100000/FPS);
        ball.move(xinfo);
        handleText(xinfo);
        repaint(xinfo);
    }
}




/*
 *   Start executing here.
 *     First initialize window.
 *     Next loop responding to events.
 *     Exit forcing window manager to clean up - cheesy, but easy.
 */
int main ( int argc, char* argv[] ) {

    if (argc == 3) {
        FPS = atoi(argv[1]);
        if (FPS == 0) FPS = 30;
        Speed = atoi(argv[2]);
        if (Speed == 0) Speed = 5;
    }

    XInfo xInfo;

    initX(argc, argv, xInfo);

    // splash screen
    splash_screen(xInfo);

    eventLoop(xInfo);

    XCloseDisplay(xInfo.display);
}
```