

```

1 /*
2  * NAME: Jiayi Wang
3  *
4  * Fall 2015 CS349 Assignment 1: An implementation of Breakout in C/C++ and Xlib.
5  *
6  *
7  *
8  * Commands to compile and run:
9  *
10 * g++ -o al al.cpp -L/usr/X11R6/lib -lX11 -lstdc++
11 * ./al
12 *
13 * Note: the -L option and -lstdc++ may not be needed on some machines.
14 */
15
16
17 #include <iostream>
18 #include <list>
19 #include <cstdlib>
20 #include <sys/time.h>
21 #include <math.h>
22 #include <stdio.h>
23 #include <unistd.h>
24 #include <vector>
25 #include <sstream>
26
27
28 /*
29  * Header files for X functions
30  */
31 #include <X11/Xlib.h>
32 #include <X11/Xutil.h>
33
34 using namespace std;
35
36 /* information of drawing, cannot change brick size */
37 const int Border = 5;
38 const int Brick_width = 80;
39 const int Brick_height = 45;
40 const int Gap = 2;
41 const int BallDiameter = 10;
42 const int Paddle_height = 5;
43
44 /* location and speed */
45 int paddle_x = 0;
46 int paddle_x_previous = 0;
47 int paddle_y = 0;
48 int ball_x = 0;
49 int ball_y = 0;
50 int Paddle_length = 100;
51
52 /* Global variables */
53 int FPS = 30; // frame per second
54 int Speed = 5; // pixel per second
55 int Score = 0;
56 int score_per_brick = 10;
57 int Window_width = 800; // init width
58 int Window_height = 600; // init height
59 const int BufferSize = 10;
60 bool gameOver = false;
61 bool gameWon = false;
62 int total_rows = 0; // rows of bricks
63 int total_cols = 0; // cols of bricks
64
65
66 /* Power-Up types */
67 int current_power_up = 0;
68 const int PU_LONG_PADDLE = 1;
69 const int PU_SHORT_PADDLE = 2;
70 const int PU_SLOW_BALL = 3;
71 const int PU_FAST_BALL = 4;
72 const int PU_DOUBLE_SCORE = 5;
73
74 /*
75  * Information to draw on the window.
76  */
77 struct XInfo {
78     Display *display;
79     int screen;
80     Window window;
81     GC gc[9];
82

```

```

83     Pixmap  pixmap;    // double buffer
84     int     width;     // size of pixmap
85     int     height;
86 };
87
88 class BrickInfo {
89 public:
90     BrickInfo(int row, int col): row(row), col(col) {
91         int random_result = rand() % 30;
92         power_up_type = (random_result < 6) ? random_result : 0;
93         if_exist = true;
94     }
95
96     int     row;
97     int     col;
98     int     power_up_type;
99     bool    if_exist;
100 };
101 std::vector<BrickInfo> BrickInfoVector;
102
103
104 /*
105  * Function to put out a message on error exits.
106  */
107 void error( string str ) {
108     cerr << str << endl;
109     exit(0);
110 }
111
112
113 /*
114  * An abstract class representing displayable things.
115  */
116 class Displayable {
117 public:
118     virtual void paint(XInfo &xinfo) = 0;
119 };
120
121
122 class Bricks : public Displayable {
123 public:
124     virtual void paint(XInfo &xinfo) {
125         for (std::vector<BrickInfo>::iterator it = BrickInfoVector.begin() ; it != BrickInfoVector.end(); ++it) {
126             BrickInfo bi = *it;
127             if (! bi.if_exist) continue;
128
129             if (bi.power_up_type != 0) { //power up
130                 XFillRectangle( xinfo.display, xinfo.pixmap, xinfo.gc[2],
131                     bi.col * (Brick_width + Gap), //x
132                     bi.row * (Brick_height + Gap), //y
133                     Brick_width, Brick_height );
134             } else {
135                 int colour_type = (bi.row % 3) + 6;
136                 XFillRectangle( xinfo.display, xinfo.pixmap, xinfo.gc[colour_type],
137                     bi.col * (Brick_width + Gap), //x
138                     bi.row * (Brick_height + Gap), //y
139                     Brick_width, Brick_height );
140             }
141         }
142     }
143
144     // constructor
145     Bricks(int null) {
146         BrickInfoVector.clear();
147
148         total_rows = 6;
149
150         total_cols = (Window_width / (Brick_width + Gap)) + 1;
151
152         for (int row = 0; row < total_rows; row++) {
153             for (int col = 0; col < total_cols; col++) {
154                 BrickInfo bi = BrickInfo(row, col);
155                 BrickInfoVector.push_back(bi);
156             }
157         }
158     }
159 };
160
161
162 class Ball : public Displayable {
163 public:
164     virtual void paint(XInfo &xinfo) {
165         XFillArc(xinfo.display, xinfo.pixmap, xinfo.gc[3], x, y, BallDiameter, BallDiameter, 0, 360*64);

```

```

166     }
167
168     void move(XInfo &xinfo) {
169
170         // hit brick
171         if ( !hit_brick ) {
172             int temp_col = x / (Brick_width + Gap);
173             int temp_row = y / (Brick_height + Gap);
174             BrickInfo bi = BrickInfoVector[temp_row * total_cols + temp_col];
175
176             if ( bi.if_exist ) {
177                 if (bi.power_up_type != 0) {
178                     current_power_up = bi.power_up_type;
179                     switch (current_power_up) {
180                         case 1: Paddle_length *= 2; break;
181                         case 2: Paddle_length /= 2; break;
182                         case 3: speed_x *= 2; speed_y *= 2; break;
183                         case 4: speed_x /= 2; speed_y /= 2; break;
184                         case 5: score_per_brick *= 2; break;
185                         default: break;
186                     }
187                 }
188
189                 speed_y = 0 - speed_y;
190                 Score += score_per_brick;
191                 bi.if_exist = false;
192                 hit_brick = true;
193             } // if ( bi.if_exist )
194
195         } else {
196             hit_brick = false;
197         }
198
199         // hit paddle
200         if ( (!hit_paddle) && ( y + BallDiameter >= paddle_y) &&
201             (y <= paddle_y+Paddle_height) && (x+BallDiameter >= paddle_x) &&
202             (x <= paddle_x+Paddle_height) ) {
203             hit_paddle = true;
204             speed_y = 0 - speed_y;
205
206             speed_x = speed_x + (paddle_x - paddle_x_previous);
207             if (speed_x > 0 && (speed_x > 1.5 * Speed)) speed_x = 1.5 * Speed;
208             if (speed_x < 0 && (speed_x < -1.5 * Speed)) speed_x = -1.5 * Speed;
209         } else {
210             hit_paddle = false;
211         }
212
213         // hit wall
214         if ( !hit_wall ) {
215
216             if (x <= 0) {
217                 speed_x = 0 - speed_x;
218                 x = 0;
219                 hit_wall = true;
220             } else if ( x + BallDiameter >= Window_width) {
221                 speed_x = 0 - speed_x;
222                 x = Window_width - BallDiameter;
223                 hit_wall = true;
224             }
225
226             if (y <= 0) {
227                 speed_y = 0 - speed_y;
228                 y = 0;
229                 hit_wall = true;
230             } else if ( y + BallDiameter >= Window_height) {
231                 hit_wall = true;
232                 gameOver = true;
233                 return;
234             }
235         } else {
236             hit_wall = false;
237         }
238
239         x = x + speed_x;
240         y = y + speed_y;
241     }
242
243     Ball(int null) {
244         x = (Window_width - BallDiameter)/2;
245         y = (Window_height/1.2)-BallDiameter;
246         speed_x = 0;
247         speed_y = Speed;
248         hit_wall = false;

```

```

249         hit_paddle = false;
250         hit_brick = false;
251     }
252
253     private:
254         int x;
255         int y;
256         int speed_x;
257         int speed_y;
258         bool hit_wall;
259         bool hit_paddle;
260         bool hit_brick;
261 };
262
263
264 class Paddle : public Displayable {
265     public:
266         virtual void paint(XInfo &xinfo) {
267             XFillRectangle( xinfo.display, xinfo.pixmap, xinfo.gc[1], paddle_x, paddle_y, Paddle_length, Paddle_height );
268         }
269
270         void moveLeft( XInfo &xinfo ) {
271             paddle_x_previous = paddle_x;
272             paddle_x = ( paddle_x - Speed < 0 ) ? 0 : paddle_x - Speed;
273         }
274
275         void moveRight( XInfo &xinfo ) {
276             paddle_x_previous = paddle_x;
277             paddle_x = ( paddle_x + Paddle_length + Speed > Window_width ) ?
278                 Window_width - Paddle_length : paddle_x + Speed - Paddle_length;
279         }
280
281         void moveAnywhere ( int x ) {
282             paddle_x_previous = paddle_x;
283
284             if ( x <= Paddle_length/2 ) {
285                 paddle_x = 0;
286             } else if ( x >= Window_width - (Paddle_length/2) ) {
287                 paddle_x = Window_width - Paddle_length;
288             } else {
289                 paddle_x = x - (Paddle_length/2);
290             }
291         }
292     }
293     // constructor
294     Paddle (int null) {
295         paddle_x = (Window_width - Paddle_length) /2;
296         paddle_y = Window_height/1.2;
297         paddle_x_previous = paddle_x;
298     }
299 };
300
301 list<Displayable*> dList;           // list of Displayables
302 Bricks bricks(0);
303 Ball ball(0);
304 Paddle paddle(0);
305
306 /*
307  * Create a window
308  */
309 void initX(int argc, char* argv[], XInfo& xInfo) {
310
311     XSizeHints hints;
312     hints.x = 100;
313     hints.y = 100;
314     hints.width = 10 * Brick_width + 9 * Gap;
315     hints.height = 6 * Brick_height + 5 * Gap + 100;
316     hints.min_width = 5 * Brick_width + 4 * Gap;
317     hints.min_height = 5 * Brick_height + 4 * Gap + 100;
318     hints.flags = PPosition | PSize;
319
320
321     /*
322     * Display opening uses the DISPLAY environment variable.
323     * It can go wrong if DISPLAY isn't set, or you don't have permission.
324     */
325     xInfo.display = XOpenDisplay( "" );
326     if ( !xInfo.display ) {
327         error( "Can't open display." );
328     }
329
330     /*
331     * Find out some things about the display you're using.

```

```

332  */
333  xInfo.screen = DefaultScreen( xInfo.display ); // macro to get default screen index
334
335  unsigned long white, black;
336  white = XWhitePixel( xInfo.display, xInfo.screen );
337  black = XBlackPixel( xInfo.display, xInfo.screen );
338
339  xInfo.window = XCreateSimpleWindow(
340      xInfo.display,           // display where window appears
341      DefaultRootWindow( xInfo.display ), // window's parent in window tree
342      hints.x, hints.y,       // upper left corner location
343      hints.width, hints.height, // size of the window
344      Border,                 // width of window's border
345      black,                  // window border colour
346      white );                // window background colour
347
348  // extra window properties like a window title
349  XSetStandardProperties(
350      xInfo.display,           // display containing the window
351      xInfo.window,           // window whose properties are set
352      "Breakout",             // window's title
353      "OW",                   // icon's title
354      None,                   // pixmap for the icon
355      argv, argc,              // applications command line args
356      &hints );                // size hints for the window
357
358  /* Colormap */
359  Colormap screen_colormap;
360  XColor red, blue, yellow, green, grey;
361  screen_colormap = DefaultColormap(xInfo.display, xInfo.screen);
362  XAllocNamedColor(xInfo.display, screen_colormap, "red", &red, &red);
363  XAllocNamedColor(xInfo.display, screen_colormap, "blue", &blue, &blue);
364  XAllocNamedColor(xInfo.display, screen_colormap, "yellow", &yellow, &yellow);
365  XAllocNamedColor(xInfo.display, screen_colormap, "green", &green, &green);
366  XAllocNamedColor(xInfo.display, screen_colormap, "grey", &grey, &grey);
367
368  /*
369   * Create Graphics Contexts
370   */
371  int i = 0;
372  xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
373  XSetForeground( xInfo.display, xInfo.gc[i], white );
374  XSetBackground( xInfo.display, xInfo.gc[i], black );
375  XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid );
376  XSetLineAttributes( xInfo.display, xInfo.gc[i],
377      1, LineSolid, CapButt, JoinRound );
378
379  // Reverse Video
380  i = 1;
381  xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
382  XSetForeground( xInfo.display, xInfo.gc[i], red.pixel );
383  XSetBackground( xInfo.display, xInfo.gc[i], white );
384  XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid );
385  XSetLineAttributes( xInfo.display, xInfo.gc[i],
386      1, LineOnOffDash, CapRound, JoinRound );
387
388  i = 2; // power up (as a red brick)
389  xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
390  XSetForeground( xInfo.display, xInfo.gc[i], red.pixel );
391  XSetBackground( xInfo.display, xInfo.gc[i], white );
392  XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid );
393  XSetLineAttributes( xInfo.display, xInfo.gc[i], Gap, LineSolid, CapRound, JoinMiter );
394
395  i = 3; // grey ball
396  xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
397  XSetForeground( xInfo.display, xInfo.gc[i], red.pixel );
398  XSetBackground( xInfo.display, xInfo.gc[i], white );
399  XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid );
400  XSetLineAttributes( xInfo.display, xInfo.gc[i], 1, LineSolid, CapRound, JoinRound );
401
402  i = 4; // black text
403  xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
404  XSetForeground( xInfo.display, xInfo.gc[i], black );
405
406  i = 5; // red text
407  xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
408  XSetForeground( xInfo.display, xInfo.gc[i], red.pixel );
409
410  i = 6; // blue brick
411  xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
412  XSetForeground( xInfo.display, xInfo.gc[i], blue.pixel );
413  XSetBackground( xInfo.display, xInfo.gc[i], white );
414  XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid );

```

```

415 XSetLineAttributes( xInfo.display, xInfo.gc[i], Gap, LineSolid, CapRound, JoinMiter );
416
417 i = 7; // yellow brick
418 xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
419 XSetForeground( xInfo.display, xInfo.gc[i], yellow.pixel );
420 XSetBackground( xInfo.display, xInfo.gc[i], white );
421 XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid );
422 XSetLineAttributes( xInfo.display, xInfo.gc[i], Gap, LineSolid, CapRound, JoinMiter );
423
424 i = 8; // green brick
425 xInfo.gc[i] = XCreateGC( xInfo.display, xInfo.window, 0, 0 );
426 XSetForeground( xInfo.display, xInfo.gc[i], green.pixel );
427 XSetBackground( xInfo.display, xInfo.gc[i], white );
428 XSetFillStyle( xInfo.display, xInfo.gc[i], FillSolid );
429 XSetLineAttributes( xInfo.display, xInfo.gc[i], Gap, LineSolid, CapRound, JoinMiter );
430
431 /* Font */
432 XFontStruct* font_info1;
433 XFontStruct* font_info2;
434 string font_name1 = "xft-xft-14-xft";
435 string font_name2 = "xft-xft-20-xft";
436 font_info1 = XLoadQueryFont( xInfo.display, font_name1.c_str() );
437 font_info2 = XLoadQueryFont( xInfo.display, font_name2.c_str() );
438 XSetFont( xInfo.display, xInfo.gc[4], font_info1->fid );
439 XSetFont( xInfo.display, xInfo.gc[5], font_info2->fid );
440
441 /* pixmap */
442 int depth = DefaultDepth( xInfo.display, DefaultScreen( xInfo.display ) );
443 xInfo.pixmap = XCreatePixmap( xInfo.display, xInfo.window, hints.width, hints.height, depth );
444 xInfo.width = hints.width;
445 xInfo.height = hints.height;
446
447
448 XSelectInput( xInfo.display, xInfo.window,
449     KeyPressMask | PointerMotionMask | StructureNotifyMask );
450
451 /*
452  * Don't paint the background -- reduce flickering
453  */
454 XSetWindowBackgroundPixmap( xInfo.display, xInfo.window, None );
455
456 /*
457  * Put the window on the screen.
458  */
459 XMapRaised( xInfo.display, xInfo.window );
460
461 XFlush( xInfo.display );
462 }
463
464 /*
465  * Function to repaint a display list
466  */
467 void repaint( XInfo &xinfo ) {
468     list<Displayable*>::const_iterator begin = dList.begin();
469     list<Displayable*>::const_iterator end = dList.end();
470
471     // draw into the buffer
472     // note that a window and a pixmap are drawable's
473     XFillRectangle( xinfo.display, xinfo.pixmap, xinfo.gc[0],
474         0, 0, xinfo.width, xinfo.height );
475     while( begin != end ) {
476         Displayable *d = *begin;
477         d->paint( xinfo ); // the displayables know about the pixmap
478         begin++;
479     }
480     // copy buffer to window
481     XCopyArea( xinfo.display, xinfo.pixmap, xinfo.window, xinfo.gc[0],
482         0, 0, xinfo.width, xinfo.height, // region of pixmap to copy
483         0, 0 ); // position to put top left corner of pixmap in window
484
485     XFlush( xinfo.display );
486
487 }
488
489 void handleKeyPress( XInfo &xinfo, XEvent &event ) {
490     KeySym key;
491     char text[BufferSize];
492
493     int i = XLookupString(
494         (XKeyEvent *)&event, // the keyboard event
495         text, // buffer when text will be written
496         BufferSize, // size of the text buffer
497         &key, // workstation-independent key symbol

```

```

498     NULL );                // pointer to a composeStatus structure (unused)
499     if ( i == 1 ) {
500         if (text[0] == 'q') {
501             error("Terminating normally.");
502             exit(0);
503         } else if (text[0] == 'c') {
504             gameWon = true;
505             Score = 10000;
506         } else if (text[0] == 'a') {
507             paddle.moveLeft(xinfo);
508         } else if (text[0] == 'd') {
509             paddle.moveRight(xinfo);
510         }
511     }
512 }
513
514 void handleMotion(XInfo &xinfo, XEvent &event) {
515     paddle.moveAnywhere(event.xbutton.x);
516 }
517
518 void handleResize(XInfo &xinfo, XEvent &event) {
519     XConfigureEvent xce = event.xconfigure;
520     fprintf(stderr, "Handling resize w=%d h=%d\n", xce.width, xce.height);
521     if (xce.width > xinfo.width || xce.height > xinfo.height) {
522         XFreePixmap(xinfo.display, xinfo.pixmap);
523         int depth = DefaultDepth(xinfo.display, DefaultScreen(xinfo.display));
524         xinfo.pixmap = XCreatePixmap(xinfo.display, xinfo.window, xce.width, xce.height, depth);
525         xinfo.width = xce.width;
526         xinfo.height = xce.height;
527     }
528 }
529
530 void handleText(XInfo &xinfo) {
531     stringstream score_ss;
532     score_ss << Score;
533     string score_string = score_ss.str();
534     XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], 10, Window_height-30, "Score: ", 7 );
535     XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], 60, Window_height-30, score_string.c_str(), score_string.length() );
536
537     stringstream FPS_ss;
538     FPS_ss << FPS;
539     string FPS_string = FPS_ss.str();
540     XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], Window_width-100, Window_height-30, "FPS: ", 5 );
541     XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], Window_width-50, Window_height-30, FPS_string.c_str(), FPS_string.length() );
542
543     string power_up_string;
544     switch (current_power_up) {
545         case 1: power_up_string = "Long paddle "; break;
546         case 2: power_up_string = "Short paddle "; break;
547         case 3: power_up_string = "Faster ball "; break;
548         case 4: power_up_string = "Slower ball "; break;
549         case 5: power_up_string = "Double score"; break;
550         default: power_up_string = "None"; break;
551     }
552     XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], 10, Window_height-10, "Power up: ", 10 );
553     XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], 80, Window_height-10, power_up_string.c_str(), power_up_string.length() );
554
555     stringstream speed_ss;
556     speed_ss << Speed;
557     string speed_string = speed_ss.str();
558     XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], Window_width-100, Window_height-10, "Ball speed: ", 12 );
559     XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[4], Window_width-70, Window_height-10, speed_string.c_str(), speed_string.length() );
560
561     if ( gameOver ) {
562         XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[5], (Window_width/2)-45, Window_height/2, "GAME OVER", 9 );
563     }
564
565     if ( gameWon ) {
566         XDrawString( xinfo.display, xinfo.pixmap, xinfo.gc[5], (Window_width/2)-45, Window_height/2, "Game WON", 8 );
567     }
568 }
569
570 }
571
572 void splash_screen ( XInfo &xinfo ) {
573     XFillRectangle(xinfo.display, xinfo.window, xinfo.gc[0], 0, 0, Window_width, Window_height);
574
575     XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 50, 20, "Breakout Game", 13 );
576     XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 50, "Name: Jiayi Wang", 16 );
577     XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 70, "ID: 20501675", 12 );
578     // blank line
579     XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 140, "- Press 's' to start the game", 29 );
580     XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 160, "- Press 'q' to quit the game", 28 );

```

```

581 XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 180, "- Press 'c' to cheat(win immediately)", 37 );
582 // blank line
583 XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 230, "You can either use your mouse,", 30 );
584 XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 250, "or press A and D on keyboard to control the paddle.", 51 );
585 // blank line
586 XDrawString( xinfo.display, xinfo.window, xinfo.gc[4], 30, 300, "Notice:", 7 );
587 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 );
588
589 XFlush( xinfo.display );
590 }
591
592 int handleStart(XInfo &xinfo, XEvent &event) {
593     KeySym key;
594     char text[BufferSize];
595
596     int i = XLookupString(
597         (XKeyEvent *)&event,    // the keyboard event
598         text,                    // buffer when text will be written
599         BufferSize,              // size of the text buffer
600         &key,                    // workstation-independent key symbol
601         NULL );                 // pointer to a composeStatus structure (unused)
602     if ( i == 1 ) {
603         if (text[0] == 's') {
604             return 1;
605         }
606     }
607     return 0;
608 }
609
610 unsigned long now() {
611     timeval tv;
612     gettimeofday(&tv, NULL);
613     return tv.tv_sec * 1000000 + tv.tv_usec;
614 }
615
616 void eventLoop(XInfo &xinfo) {
617     // Add stuff to paint to the display list
618     dList.push_front(&bricks);
619     dList.push_front(&ball);
620     dList.push_front(&paddle);
621
622     XEvent event;
623     unsigned long lastRepaint = 0;
624
625     while(true) {
626         int ret_val = 0;
627         if (XPending(xinfo.display) > 0) {
628             XNextEvent( xinfo.display, &event );
629             if( event.type == KeyPress) ret_val = handleStart(xinfo, event);
630         }
631         if (ret_val == 1) break;
632     }
633
634     while( !gameOver && !gameWon ) {
635         if (XPending(xinfo.display) > 0) {
636             XNextEvent( xinfo.display, &event );
637             switch( event.type ) {
638                 case KeyPress:
639                     handleKeyPress(xinfo, event);
640                     break;
641                 case MotionNotify:
642                     handleMotion(xinfo, event);
643                     break;
644                 case ConfigureNotify:
645                     handleResize(xinfo, event);
646                     break;
647             }
648         }
649
650         usleep(100000/FPS);
651         ball.move(xinfo);
652         handleText(xinfo);
653         repaint(xinfo);
654     }
655 }
656
657
658
659
660 /*
661 * Start executing here.
662 * First initialize window.
663 * Next loop responding to events.

```



```
664 *      Exit forcing window manager to clean up - cheesy, but easy.
665 */
666 int main ( int argc, char* argv[] ) {
667
668     if (argc == 3) {
669         FPS = atoi(argv[1]);
670         if (FPS == 0) FPS = 30;
671         Speed = atoi(argv[2]);
672         if (Speed == 0) Speed = 5;
673     }
674
675     XInfo xInfo;
676
677     initX(argc, argv, xInfo);
678
679     // splash screen
680     splash_screen(xInfo);
681
682     eventLoop(xInfo);
683
684     XCloseDisplay(xInfo.display);
685 }
```