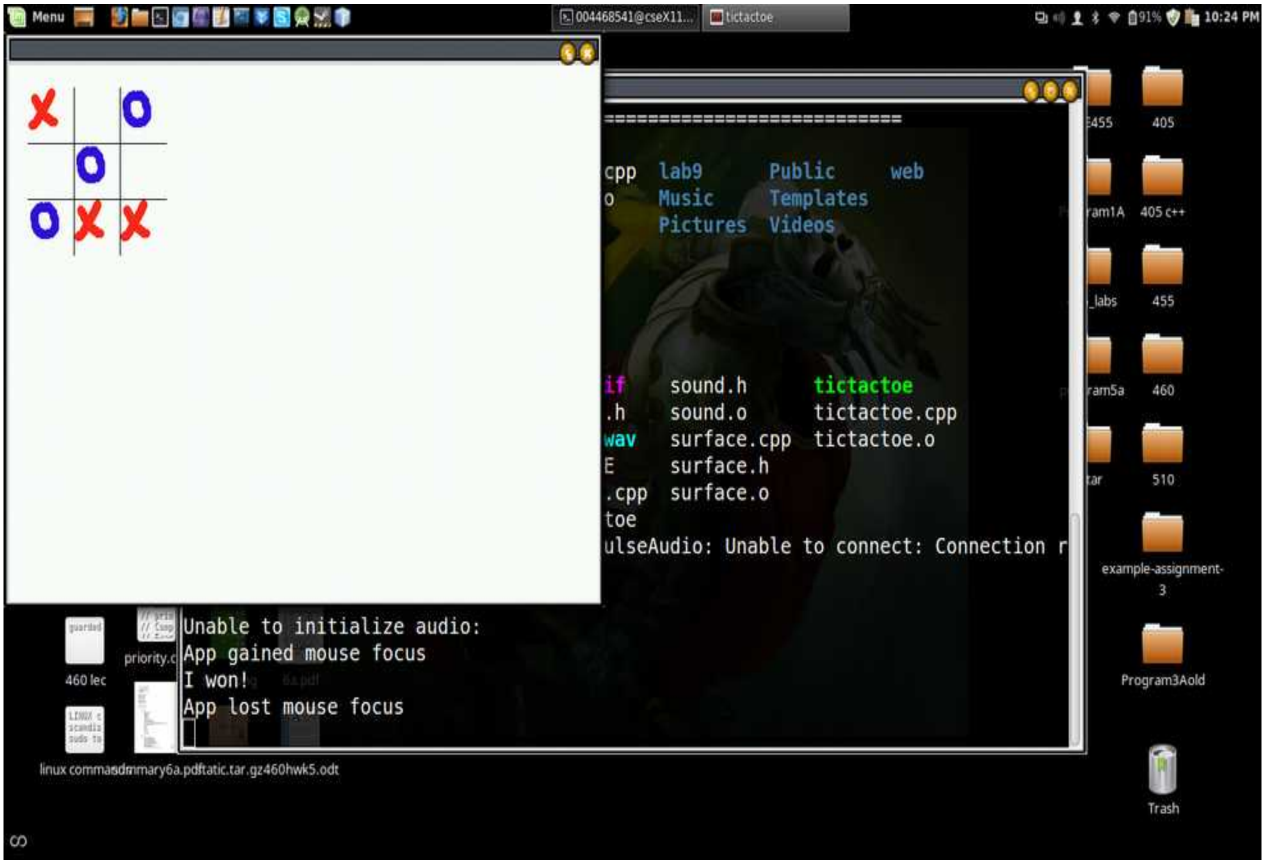
Dean Cosanella
Flavio dos Santos-Ross
CSE 460
Dr. Tong Lai Yu
Lab 9 Report

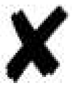
For this lab, we were to download the tictactoe.tgz file given in the lab and compile the code given in that file, test it, and modify it to allow for two players to play the game as well as changing the graphics and audio that are used when the game is being played. We first played the game a couple of times before we started making changes and one of our games ended up in the following result:

Screenshot of original tic-tac-toe game:



```
===========================
cpp  lab9      Public     web
o    Music     Templates
     Pictures  Videos

if    sound.h      tictactoe
.h    sound.o      tictactoe.cpp
wav   surface.cpp  tictactoe.o
E     surface.h
.cpp  surface.o
toe
ulseAudio: Unable to connect: Connection r
Unable to initialize audio:
App gained mouse focus
I won!
App lost mouse focus
```

Once we played the game a couple of times, we searched the Internet and found the following images to use as our new X and O:



We also changed the audio files and gave them the same names as the original files so we would not have to change the code for those files.

To make this a two-player game, we modified the game_thread.cpp file. We first added a boolean value “player” so we would be able to switch between player 1 and player 2. We then looked at the game_thread which is the logic of the game. We changed it to allow the players to exchange moves by using the conditions if (player == false) and if (player == true) respectively. After the code finished with the logic within these two statements, we made player = !player until the game was over. This allowed both players to make moves. The modified game_thread.cpp file is shown below:

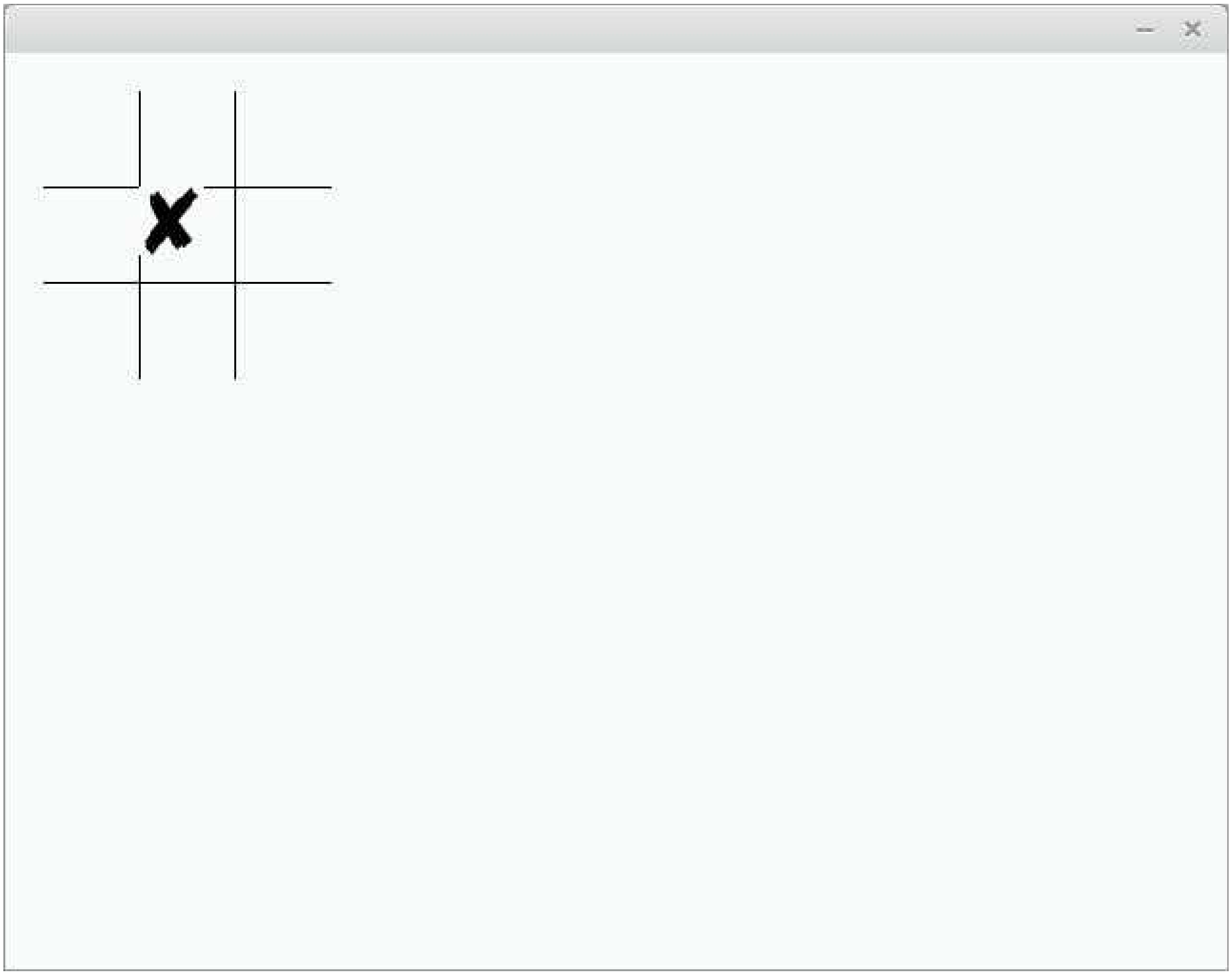
Code for game_thread.cpp:

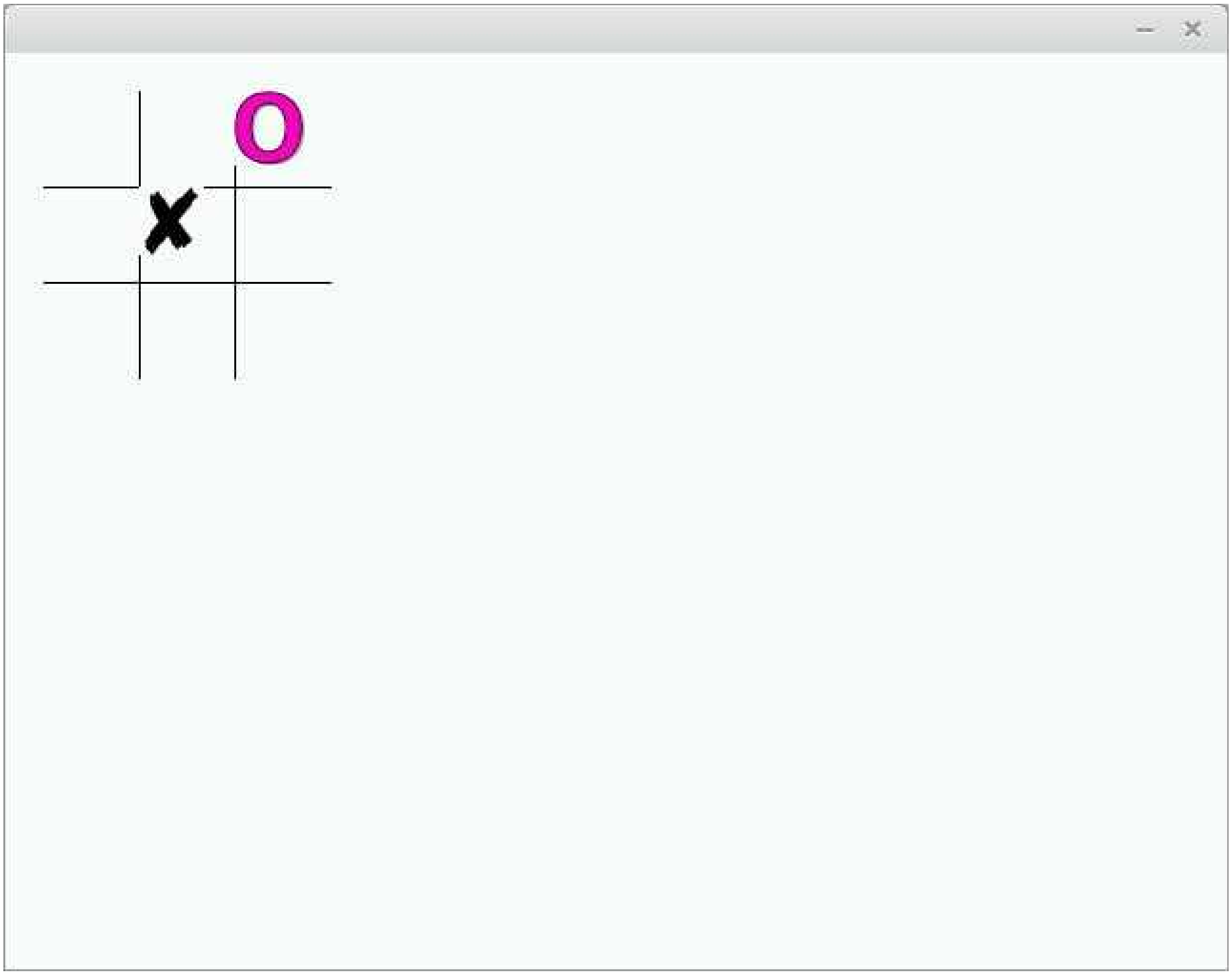
```
/*
game_thread.cpp -- thread functions for playing the game
Four lines are drawn to form the figure #.
The quadrants are labeled from 0 to 8 with the upper left
corner labeled as 0, the next column as 1; next row starts at 3.
*/
#include "SDL/SDL.h"
#include "SDL/SDL_image.h"
#include "SDL/SDL_mixer.h"
#include "SDL/SDL_thread.h"
#include <string>
#include <iostream>
#include <stdlib.h>
#include <vector>               //STL library
#include <deque>                //STL Library
#include "surface.h"
#include "point.h"
#include "sound.h"

using namespace std;

extern deque<char>keyq;         //a queue to save values of keys pressed
extern deque<Point>mouseq;      //queue to save mouse coordinates
extern deque<char>numq;         //queue to save digits
extern bool quit;               //global variable to quit game
extern SDL_mutex *key_mutex;    //mutex to lock key variables
extern SDL_mutex *mouse_mutex;  //mutext to lock mouse variables

bool game_over = false;         //shared with sound_thread

bool player = false;

//parameters for figure #
const int x0 = 20, y0 = 20;
const int dx = 50, dy = 50, len = 150;
const int xLEFT = x0 + dx;
const int xRIGHT = x0 + 2*dx;
const int yTOP = y0 + dy;
const int yBOTTOM = y0 + 2*dy;

//centers for drawing crosses or circles on #
```

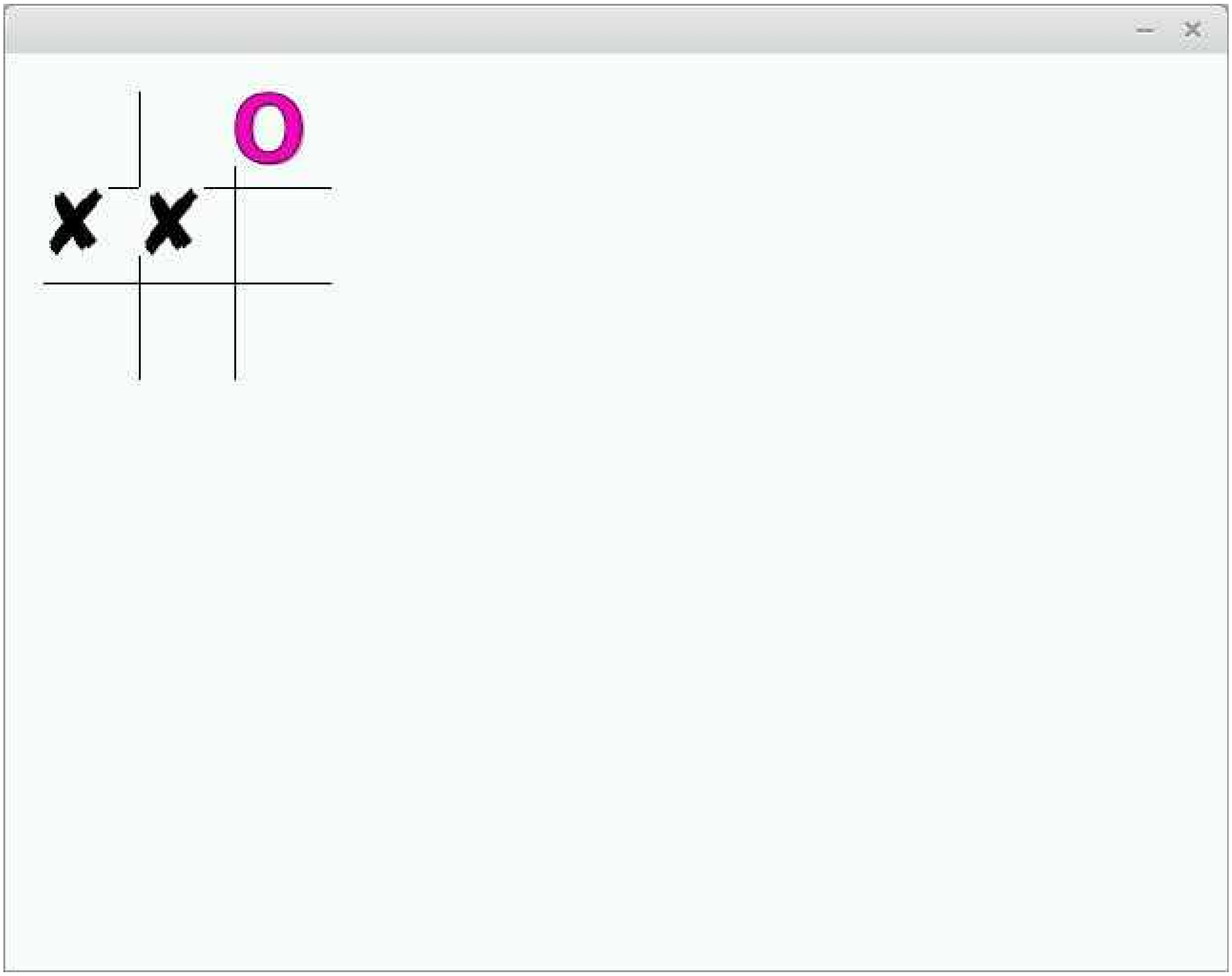
```
Point draw_centers[9] = {
    Point ( x0,  y0  ), Point(  x0 + dx, y0  ), Point ( x0 + 2*dx, y0  ),
    Point ( x0,  y0+dy ), Point(  x0+dx, y0 + dy ), Point ( x0 + 2*dx, y0 + dy ),
    Point ( x0,  y0+2*dy ), Point(  x0+dx, y0 + 2*dy ), Point ( x0+2*dx, y0 +2*dy )
};

/*
  Given 'start' and 'x', find the quadrant number.
  The first row starts at 0, second row at 3 and third row at 6.
*/
int check_x ( int start, int x )
{
  if ( x <= xLEFT )
    return start;
  else if ( x <= xRIGHT )
    return start + 1;
  else
    return start + 2;

  return 0;
}

//check which quadrant the point is in
int check_quadrant ( Point p )
{
  if ( p.y <= yTOP )            //in either quadrant 0, 1, or 2
    return  check_x ( 0, p.x );
  else if ( p.y <= yBOTTOM )    //in either 3, 4, or 5
    return check_x ( 3, p.x );
  else                          //in either 7, 7, or 8
    return check_x ( 6, p.x );
}

//load an image on 'screen' at 'center'
int load_image (   SDL_Surface *screen, char *image_name, Point center )
{
  SDL_Surface *image;
  SDL_Rect source, offset;

  image = IMG_Load(  image_name );
  if ( image == NULL ) {
    cout << "Unable to load image\n";
    return 1;
  }
  source.x = 0;
  source.y = 0;
  source.w = image->w;
  source.h = image->h;
  offset.x = center.x;
  offset.y = center.y;
  offset.w = image->w;
  offset.h = image->h;
  SDL_BlitSurface ( image, &source, screen, &offset );
}

/*
 * Some of the following are copied from Sun's Java Tic Tac Toe demo program
 * In this game a position is represented by a white and black
 * bitmask. A bit is set if a position is ocupied. There are
 * 9 squares so there are 1<<9 possible positions for each
 * side. An array of 1<<9 booleans is created, it marks
 * all the winning positions.
*/

/**
 * White's current position. The computer is white.
 */
int white;

/**
 * Black's current position. The user is black.
 */
int black;

/**
 * The squares in order of importance...
 */
const int moves[] = {4, 0, 2, 6, 8, 1, 3, 5, 7};

/**
 * The winning positions.
 */
bool won[512];
const int DONE = (1 << 9) - 1;
const int OK = 0;
const int WIN = 1;
const int LOSE = 2;
const int STALEMATE = 3;

/**
 * Mark all positions with these bits set as winning.
 */
void isWon(int pos) {
  for (int i = 0 ; i < DONE ; i++) {
    if ((i & pos) == pos) {
      won[i] = true;
    }
   }
}

/**
 * Initialize all winning positions.
 */
void init_win () {
  for ( int i = 0; i <= DONE; i++ )    //reset all positions
    won[i] = false;
  isWon((1 << 0) | (1 << 1) | (1 << 2));
  isWon((1 << 3) | (1 << 4) | (1 << 5));
  isWon((1 << 6) | (1 << 7) | (1 << 8));
  isWon((1 << 0) | (1 << 3) | (1 << 6));
  isWon((1 << 1) | (1 << 4) | (1 << 7));
  isWon((1 << 2) | (1 << 5) | (1 << 8));
  isWon((1 << 0) | (1 << 4) | (1 << 8));
  isWon((1 << 2) | (1 << 4) | (1 << 6));
}

/**
 * Compute the best move for white.
 * return the square to take
 */
int bestMove(int white, int black) {
  int bestmove = -1;

loop:
  for (int i = 0 ; i < 9 ; i++) {
    bool continue1 = false;
    int mw = moves[i];
    if (((white & (1 << mw)) == 0) && ((black & (1 << mw)) == 0)) {
      int pw = white | (1 << mw);
      if (won[pw]) {
        // white wins, take it!
        return mw;
      }
      for (int mb = 0 ; mb < 9 ; mb++) {
        if (((pw & (1 << mb)) == 0) && ((black & (1 << mb)) == 0)) {
          int pb = black | (1 << mb);
          if (won[pb]) {
            // black wins, take another
            //continue loop;
            continue1 = true;
            continue;
          }
        } //if pw
      }// for mb
      if ( continue1 ) continue;
      // Neither white nor black can win in one move, this will do.
      if (bestmove == -1) {
        bestmove = mw;
      }
    }
  } //for i
  if (bestmove != -1) {
   return bestmove;
  }

  // No move is totally satisfactory, try the first one that is open
  for (int i = 0 ; i < 9 ; i++) {
   int mw = moves[i];
   if (((white & (1 << mw)) == 0) && ((black & (1 << mw)) == 0)) {
    return mw;
   }
  }
  // No more moves
  return -1;
} //bestMove

/**
 * User 1 move.
 * return true if legal
 */
bool yourMove(int m) {
  if ((m < 0) || (m > 8)) {
    return false;
  }
  if (((black | white) & (1 << m)) != 0) {
    return false;
  }
  black |= 1 << m;
  return true;
}

/**
 * User 2 move.
 * return true if legal
 */
bool myMove(int m) {
  if ((m < 0) || (m > 8)) {
    return false;
  }
  if (((black | white) & (1 << m)) != 0) {
    return false;
  }
  white |= 1 << m;
  return true;
}

/**
 * Figure what the status of the game is.
 */
 int status() {
   if (won[white]) {
     return WIN;
  }
  if (won[black]) {
     return LOSE;
  }
  if ((black | white) == DONE) {
    return STALEMATE;
  }
  return OK;
}

/**
 * Who goes first in the next game?
 */

bool first = true;      //set to true again in init_board()
bool occupied[9];       //indicates if a quadrant has been marked

void init_board( Surface *surf )
{
  int x, y;

  //clear the screen
  surf->clearScreen();

  x = x0;
  y = y0 + dy;
  surf->setColor ( 0, 0, 0 );   //draw lines in black
  surf->moveTo ( x, y );
  surf->lineTo ( x + len, y );  //draw upper horizontal line
  x = x0;
  y += dy;
  surf->moveTo ( x, y );
  surf->lineTo ( x + len, y );  //draw lower horizontal line
  x = x0 + dx;
  y = y0;
  surf->moveTo ( x, y );        //draw left vertical line
  surf->lineTo ( x, y + len );
  x = x0 + 2*dx;
  y = y0;
  surf->moveTo ( x, y );
  surf->lineTo ( x, y + len );  //draw right vertical line

  white = black = 0;
  //initialize the winning positions
  init_win();
  for ( int i = 0; i < 9; ++i )
    occupied[i] = false;        //no square is occupied

  //clear the mouse queue
   SDL_mutexP ( mouse_mutex );          //lock before accessing mouseq
   while ( mouseq.size() )
      mouseq.pop_front();
   SDL_mutexV ( mouse_mutex );          //release lock

   //clear the num queue, key queue
   SDL_mutexP ( key_mutex );            //lock before clear queues
   while ( numq.size() )
      numq.pop_front();
   while ( keyq.size() )
      keyq.pop_front();
   SDL_mutexV ( key_mutex );            //release lock
}


/*
  This it the thread that plays the game.
*/
int game_thread ( void *surface )
{
  Point p;
  int q;

  Surface *surf = ( Surface * ) surface;
  SDL_Surface *screen = ( SDL_Surface *) surf->getSurface();

  //initialize game board and prepare to play
  init_board( surf );

    while ( !quit ) {
    if ( mouseq.size() > 0 || numq.size() > 0 ){ //either mouse clicked or num key
pressed
      if ( mouseq.size() > 0 ) {        //mouse has been clicked
        SDL_mutexP ( mouse_mutex );     //lock before accessing mouse queue
        p = mouseq.front();             //get point from front of mouse queue
        mouseq.pop_front();             //remove point from queue
        SDL_mutexV ( mouse_mutex );     //release lock
        q = check_quadrant( p );        //check which quadrant has been clicked
      } else {                          //num key has been pressed
    SDL_mutexP ( key_mutex );   //lock before accessing num queue
    q = ( int ) numq.front() - '0';
    numq.pop_front();
    SDL_mutexV ( key_mutex );   //release lock
      }
      if ( occupied[q] ) continue;      //don't do anything if quadrant arleady
taken
      switch (( status() ) ) {          //check status
      case WIN:
      case LOSE:
      case STALEMATE:
        white = black = 0;
        if (first)
            white |= 1 << (int)( random() % 9);
        first = !first;
        continue;
      }  //switch

      if (player == false) {
        load_image ( ( SDL_Surface *) screen, "cross.gif", draw_centers[q] );
        occupied[q] = true;

        if ( yourMove ( q ) ) {
            switch ( status() ) {
                case WIN:
                    cout << "Player 2 won!" << endl;
                    game_over = true;
                    play_sfx ( "end.wav" );             //play music to celebrate
win
                break;
                case LOSE:
                    cout << "Player 1 won!" << endl;
                    game_over = true;
                    play_sfx ( "gameover.wav" );        //play music to signal loss
                break;
                case STALEMATE:
                    cout << "Its a tie!" << endl;
                    game_over = true;
                    play_sfx ( "quit.wav" );            //play music to signal
STALEMATE
                break;
            } //switch
        }
    } else if (player == true) {
        load_image ( ( SDL_Surface *) screen, "not.gif", draw_centers[q] );
        occupied[q] = true;

        if ( myMove ( q ) ) {
            switch ( status() ) {
                case WIN:
                    cout << "Player 2 won!" << endl;
                    game_over = true;
                    play_sfx ( "end.wav" );             //play music to celebrate
win
                break;
                case LOSE:
                    cout << "Player 1 won!" << endl;
                    game_over = true;
                    play_sfx ( "gameover.wav" );        //play music to signal loss
                break;
                case STALEMATE:
                    cout << "Its a tie!" << endl;
                    game_over = true;
                    play_sfx ( "quit.wav" );            //play music to signal
STALEMATE
                break;
            } //switch
      }
    } //inner switch

    player = !player; // Switch Players

    } // if ( mouseq.size() > 0 )
    while ( game_over ) {
    while ( !keyq.size() && !quit )     //waiting for key input
      SDL_Delay( 5 );
    if ( quit ) break;          //all done
    if ( keyq.front() == 's' || keyq.front() == 'S' ) {
      init_board( surf );               //start over again
      game_over = false;
    }
    } //while ( game_over )
    SDL_Delay ( 5);                     //give up some cpu time for others
    } //while ( !quit )         //quit may be set to true in key_thread or
mouse_thread
    return 0;
}
```

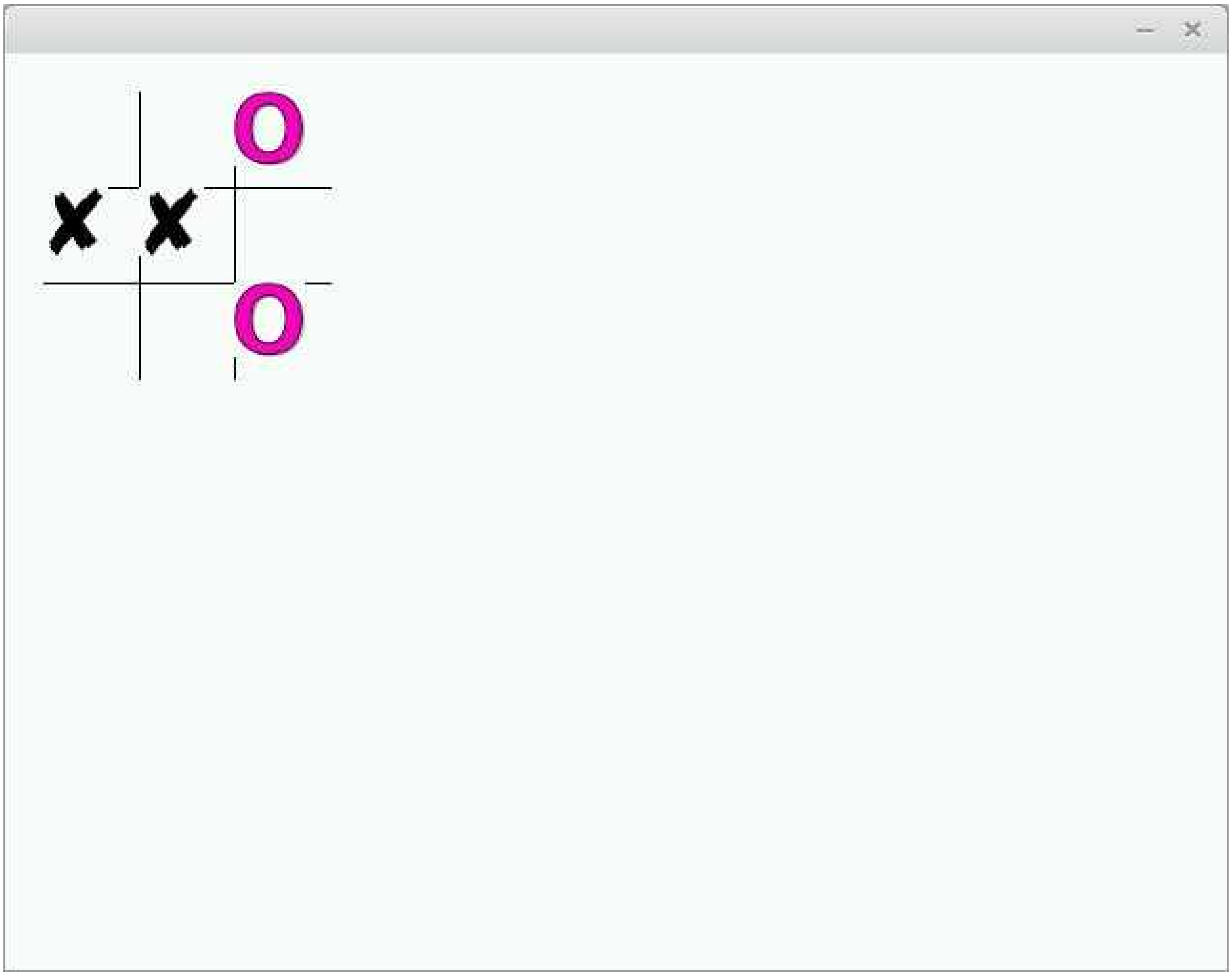
The rest of this lab is continued on the next page.

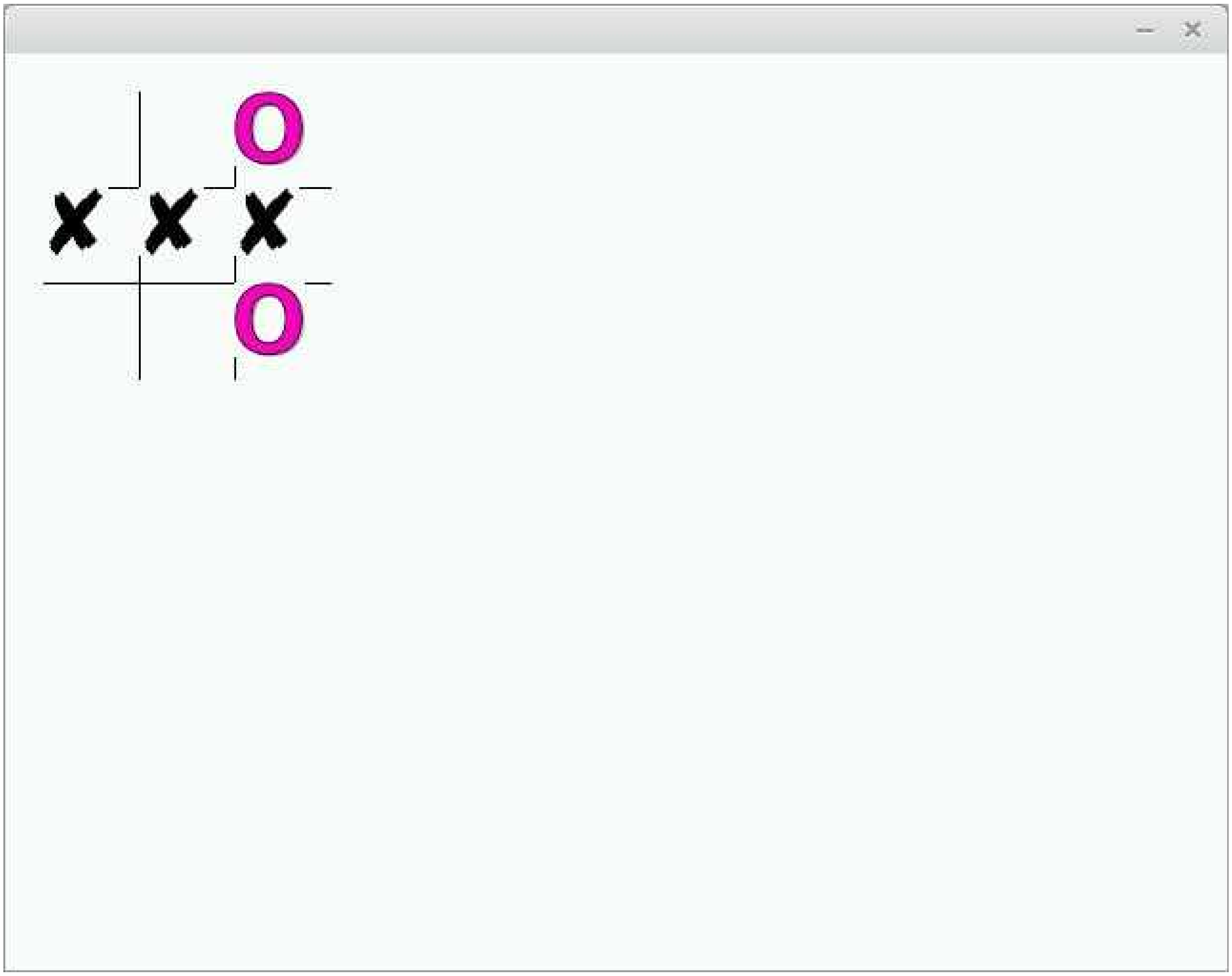
After we recompiled the entire program, we get the following results:




O
X


O
X X


O
X X
O



As you can see from the five screenshots above, this modified tic-tac-toe game allows two players to play and input their X or O into any open space. The only problem we had was making the images centered in their respective locations. Other than that the game works perfectly.

This lab taught us a lot about using SDL threads in simple games such as tic-tac-toe. It is very interesting to see how these different threads work together to create a game with graphics, sound, and user input. Because we were able to fully complete this lab, we give ourselves a 20/20 on this lab.