```
// generated by Fast Light User Interface Designer (fluid) version 1.0300

#include "TZH.h"
/**
```

1 Specification

In this post-apocalyptic experience, a lone hero fights for survival against endless waves of zombies in order to protect his one true love: ham.

```
*/
//
/**
```

2 Analysis

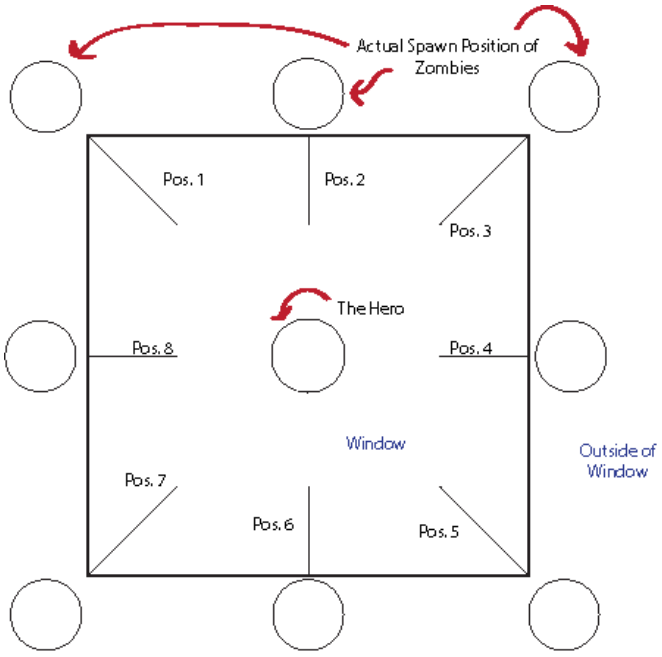
2.1 Inputs

Arrow Keys: The user can use the left and right keys for turning the hero.

Space Bar: The user hits the space bar to make the hero shoot.

2.2 Process

The program will open up with the WAT Interactive logo and intro music for 2 seconds. After that, we will then skip forward to the game's main menu. In the main menu, the program will provide the main game screen.



The game screen will consist of 8 points surrounding the center. A random number generator will be used to set spawn rates and spawn points. The results of the random number generator will determine which of the 8 markers will spawn a zombie that will attack a hero at the center. Each zombie will move at an increasing speed as time progresses and as the user manages to overcome his or her adversaries.

The hero will be stationary in the center and be rotated by the user using arrow keys. The space bar will be mapped to an event that will shoot out bullets from the hero's gun, thus allowing the user to defend him or herself from the horde.

The bullets shot by the hero will either collide with a zombie or miss. If the bullet misses, nothing happens. If the program detects that the bullet collides with the zombie's visible body, the zombie will display a death animation and its box will be retrieved to its starting position, ready to deploy once more when the random number generator selects it.

If a zombie manages to get to the hero or the beloved ham, the game will end. A cutscene will play, showing the inevitable aftermath of the hero's loss, and the player will be redirec to the main menu.

2.3 Outputs

Background Audio

SFX: Appropriate audio will accompany gunfire, dying, and defeating a zombie.

Fun!

*/
//
/**

3 Design

3.1 Prototype

Play an intro.

Display the main menu.

When the user selects the "start" button, go to the main game screen.

While the hero is still alive:

- Spawn a zombie.
- Set the zombie's speed based on how many zombies have been killed by this point.
- Allow the hero to move and shoot.
- If a bullet collides with a zombie, add 1 to the zombie count.
- If a zombie collides with the hero or the ham, set the hero to dead.

3.2 Final

Define a function for handling the user's inputs.

- If the user hits the right key, turn the hero clockwise.
- If the user hits the left key, turn the hero counterclockwise.
- When the user hits the space bar, fire a projectile.

Set the zombies to their default positions outside of the window.

Play the introduction by loading images into an array and display via looping.

Display the main menu with QUIT, INSTRUCTIONS, and START options.

QUIT: Displays a confirmation screen to either cancel the quit or quit.
INSTRUCTIONS: Displays an image that shows the user how to play.
START: Loads the game screen.
CREDITS: Shows the credits of everyone who worked on the game.

In the game screen:

- Spawn a zombie in one out of four or eight locations.
- Move the zombie forward.
- If a bullet collides with the zombie, add one to a zombie count and reset the zombie.
- If a zombie collides with the hero, play a cutscene and go to the game over screen.

Display the game over menu with QUIT, REPLAY, MAIN MENU, and CREDITS options.

QUIT: Displays a confirmation screen to either cancel the quit or quit.
REPLAY: Reloads the game screen and lets the user try again.
MAIN MENU: Returns the user to the main menu.

```
*/
//
/**
```

4 Implementation

```
*/
//
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_PNG_Image.H>
#include <FL/Fl_JPEG_Image.H>
#include <cstdlib>
#include <ctime>
```

```
#include <windows.h>
#include <sstream>
#include <iostream>
#include "audiere.h"
using namespace std;
static const int LOGO_FRAMES = 34;
static const int TOP_DEATH_FRAMES = 13;
static const int CREDITS_FRAMES = 35;
static const int ZOMBIE_RIGHT_DEAD_FRAMES = 4;
static const int ZOMBIE_LEFT_DEAD_FRAMES = 4;
static const int ZOMBIE_UP_DEAD_FRAMES = 4;
static const int ZOMBIE_DOWN_DEAD_FRAMES = 4;
static const int SNOOP_FRAMES = 58;
static Fl_PNG_Image* snoopa[SNOOP_FRAMES];
static Fl_PNG_Image* logo[LOGO_FRAMES];
static Fl_PNG_Image* topdeath[TOP_DEATH_FRAMES];
static Fl_PNG_Image* creditsa[CREDITS_FRAMES];
static Fl_PNG_Image* right_zombied[ZOMBIE_RIGHT_DEAD_FRAMES];
static Fl_PNG_Image* left_zombied[ZOMBIE_LEFT_DEAD_FRAMES];
static Fl_PNG_Image* up_zombied[ZOMBIE_UP_DEAD_FRAMES];
static Fl_PNG_Image* down_zombied[ZOMBIE_DOWN_DEAD_FRAMES];
using namespace audiere;
static AudioDevicePtr device(OpenDevice());
static OutputStreamPtr speedupsound = OpenSound(device, "SANIC.mp3");
static OutputStreamPtr threeloudfiveme = OpenSound(device, "twoloudfouru.wav");
static OutputStreamPtr menumusic = OpenSound(device, "MenuMusicTrim.wav");
static OutputStreamPtr gamemusic = OpenSound(device, "gamemusic.mp3");
static OutputStreamPtr gunshot = OpenSound(device, "barreta_m9.wav");
static OutputStreamPtr deathmusic = OpenSound(device, "diemusic.wav");
static OutputStreamPtr replaymusic = OpenSound(device, "revivemusic.wav");
static OutputStreamPtr creditmusic = OpenSound(device, "creditmusic.mp3");
/**
   this is for the rate at which the zombies spawn.
*/
static double Spawn_Variable = 1;
/**
   for checking if the game is on
*/
static int gameoncheck = 0;
/**
   This is a boolean variable for checking if a zombie is still on the screen.
*/
static int rightzombie = 0;
/**
   This is a boolean variable for checking if a zombie is still on the screen.
*/
static int leftzombie = 0;
/**
   This is a boolean variable for checking if a zombie is still on the screen.
*/
```

```
static int upzombie = 0;
/**
   This is a boolean variable for checking if a zombie is still on the screen.
*/
static int downzombie = 0;
static int keytotal = 40000;
/**
   For the function in select to prevent duplicate spawns.
*/
static int lastnum = 0;
/**
   For the function in select to prevent trips.
*/
static int prevlastnum = 0;
/**
   Sets the frame rate for sprites that move, for use in repeat_timeout functions.
*/
static double SpriteFrameRate = 0.02;
/**
   Sets the speed of the zombies.
*/
static int ZombieSpeed = 1;
/**
   Sets the speed of the bullets.
*/
static int BulletSpeed = 8;
/**
   This keeps track of how many zombies have died. The value of this
   globally-declared variable will be used to impact several different
   functions within the program.
*/
static double zombiecount = 0;

/**
   integers are box position
*/
SoundBox::SoundBox(int x, int y, int w, int h):Fl_Box(x, y, w, h) {
  /** Code for audiere play sound

  Commented out to make global, not sure if this is needed in the
  class anymore, leaving in code just to make sure.

  OutputStreamPtr s = OpenSound(device, sound_name.c_str());
  s->play();

  Note from Brian M.:
  This code may or may not prove to be needed in our final, but it can
  provide a useful resource for whenever we need to play a sound, and from
  my knowledge, we might need to play sounds a lot more than twice if we want
  our game to have any sort of sound effects.
```

```
  */
}

/**
   This is for tracking what the user does.
*/
int SoundBox::handle(int e) {
  //cout << "in handle" << endl;

  int r = 0;
  /*
  static int xx = 0;
  static int yy = 0;
  */

  //OutputStreamPtr sound = OpenSound(device, sound_name.c_str());

  switch (e)
  {

        case FL_FOCUS:
                //cout << "Within FL_FOCUS" << endl;
                r = 1;
                break;

        case FL_KEYDOWN: //detects keystrokes

                //cout << "button pressed" << endl;
                        int key = Fl::event_key(); //doing math
                        //int keytotal = 40000; (changed to global var. for implementing)

                        if (key == FL_Left) //player hit left key
                                keytotal--; //so keytotal goes down.
                        else if (key == FL_Right) //player hit right key
                                keytotal++; //so keytotal goes up.

                        int decision = keytotal % 4; //modulus for directions

                        if(key == 32){ //If the player hit the space bar...
                                gunfire(decision); //Pass decision into gunfire
                        }
                        // cout << decision << endl; // ending math

                        switch (decision) //displays player image based on math above
                        {
                                case 0: // pointing right
                                player_facing(RIGHT);
                                break;
```

```
                                case 1: // pointing down
                                player_facing(DOWN);
                                break;

                                case 2: // pointing left
                                player_facing(LEFT);
                                break;

                                case 3: // pointing up
                                player_facing(UP);
                                break;
                        }
                break; //This break corresponds to FL_KEYDOWN
        }
  return r;
}

Fl_Double_Window *quit_confirm=(Fl_Double_Window *)0;

Fl_Box *QuitBackground=(Fl_Box *)0;

Fl_Button *yes=(Fl_Button *)0;

static void cb_yes(Fl_Button*, void*) {
  cout << "Thanks for playing!" << endl;
quit_confirm->hide();
window->hide();
scorewin->hide();
}

Fl_Button *no=(Fl_Button *)0;

static void cb_no(Fl_Button*, void*) {
  quit_confirm->hide();
}

Fl_Double_Window *scorewin=(Fl_Double_Window *)0;

Fl_Output *scoreout=(Fl_Output *)0;

Fl_Double_Window *window=(Fl_Double_Window *)0;

Fl_Text_Display *GameOver=(Fl_Text_Display *)0;

Fl_Box *death=(Fl_Box *)0;

Fl_Box *gameoverbackground=(Fl_Box *)0;

Fl_Box *background=(Fl_Box *)0;
```

```
Fl_Button *start=(Fl_Button *)0;

static void cb_start(Fl_Button*, void*) {
  set_gamescreen(1,0); //Set up the main game.;
}

Fl_Button *quit=(Fl_Button *)0;

static void cb_quit(Fl_Button*, void*) {
  quit_confirm->show(); // (show a confirmation window).
QuitBackground->image(new Fl_PNG_Image("InstructionsBW.png"));
QuitBackground->parent()->redraw();
}

Fl_Box *map=(Fl_Box *)0;

Fl_Box *zombie_right=(Fl_Box *)0;

Fl_Box *zombie_left=(Fl_Box *)0;

Fl_Box *zombie_up=(Fl_Box *)0;

Fl_Box *zombie_down=(Fl_Box *)0;

Fl_Box *Ham=(Fl_Box *)0;

Fl_Box *player=(Fl_Box *)0;

Fl_Box *bullet_right=(Fl_Box *)0;

static void cb_bullet_right(Fl_Box* o, void*) {
  //o->image(new Fl_PNG_Image("Bullet_right.png"));
}

Fl_Box *bullet_left=(Fl_Box *)0;

static void cb_bullet_left(Fl_Box* o, void*) {
  //o->image(new Fl_PNG_Image("Bullet_left.png"));
}

Fl_Box *bullet_up=(Fl_Box *)0;

static void cb_bullet_up(Fl_Box* o, void*) {
  //o->image(new Fl_PNG_Image("Bullet_up.png"));
}

Fl_Box *bullet_down=(Fl_Box *)0;

SoundBox *music=(SoundBox *)0;
```

```
Fl_Button *replay=(Fl_Button *)0;

static void cb_replay(Fl_Button*, void*) {
  replaymusic->play();
Fl::add_timeout(0.001, top_death_fadeout);
}

Fl_Button *credit_button=(Fl_Button *)0;

static void cb_credit_button(Fl_Button*, void*) {
  menumusic->stop();
creditmusic->play();
creditmusic->setRepeat(true);
Fl::add_timeout(0.001, display_credits);
}

Fl_Box *title=(Fl_Box *)0;

Fl_Box *instructions=(Fl_Box *)0;

Fl_Button *instr_button=(Fl_Button *)0;

static void cb_instr_button(Fl_Button*, void*) {
  Fl::add_timeout(0.01, display_instr);
}

Fl_Box *Credits=(Fl_Box *)0;

Fl_Button *menu_button=(Fl_Button *)0;

static void cb_menu_button(Fl_Button*, void*) {
  Fl::add_timeout(0.01, back_to_menu);
}

Fl_Button *x2loud4u=(Fl_Button *)0;

static void cb_x2loud4u(Fl_Button*, void*) {
  creditmusic->stop();
threeloudfiveme->play();
}

Fl_Box *speedupbox=(Fl_Box *)0;

Fl_Box *snoopbox=(Fl_Box *)0;

int main(int argc, char **argv) {
  { quit_confirm = new Fl_Double_Window(520, 430, "Confirmation");
    quit_confirm->color(FL_FOREGROUND_COLOR);
    { QuitBackground = new Fl_Box(0, 0, 520, 430);
    } // Fl_Box* QuitBackground
```

```
  { Fl_Box* o = new Fl_Box(20, 72, 480, 43, "DO YOU REALLY WANT TO QUIT?");
    o->labelsize(29);
    o->labelcolor((Fl_Color)1);
  } // Fl_Box* o
  { yes = new Fl_Button(128, 282, 245, 45, "Yes, I\'m done.");
    yes->box(FL_NO_BOX);
    yes->color(FL_FOREGROUND_COLOR);
    yes->labelsize(38);
    yes->labelcolor(FL_BACKGROUND2_COLOR);
    yes->callback((Fl_Callback*)cb_yes);
  } // Fl_Button* yes
  { no = new Fl_Button(43, 176, 435, 55, "No, I must save the ham!");
    no->box(FL_NO_BOX);
    no->color(FL_FOREGROUND_COLOR);
    no->labelsize(38);
    no->labelcolor((Fl_Color)1);
    no->callback((Fl_Callback*)cb_no);
  } // Fl_Button* no
  quit_confirm->end();
} // Fl_Double_Window* quit_confirm
{ scorewin = new Fl_Double_Window(164, 51, "Score");
  scorewin->color(FL_FOREGROUND_COLOR);
  { scoreout = new Fl_Output(0, 5, 150, 40);
    scoreout->box(FL_NO_BOX);
    scoreout->color((Fl_Color)1);
    scoreout->textsize(28);
    scoreout->textcolor((Fl_Color)1);
    scoreout->align(Fl_Align(FL_ALIGN_RIGHT));
  } // Fl_Output* scoreout
  scorewin->end();
} // Fl_Double_Window* scorewin
{ window = new Fl_Double_Window(720, 450, "Top Zombie Ham");
  window->color(FL_FOREGROUND_COLOR);
  { GameOver = new Fl_Text_Display(360, 210, 0, 0, "GAME OVER");
    GameOver->box(FL_NO_BOX);
    GameOver->labelsize(71);
    GameOver->labelcolor((Fl_Color)1);
  } // Fl_Text_Display* GameOver
  { death = new Fl_Box(0, 25, 720, 405);
    death->box(FL_FLAT_BOX);
    death->color((Fl_Color)67);
  } // Fl_Box* death
  { gameoverbackground = new Fl_Box(10, 18, 720, 450);
  } // Fl_Box* gameoverbackground
  { background = new Fl_Box(0, 0, 725, 450);
  } // Fl_Box* background
  { start = new Fl_Button(535, 80, 140, 30, "START");
    start->box(FL_NO_BOX);
    start->color((Fl_Color)128);
    start->labelsize(41);
```

```
  start->callback((Fl_Callback*)cb_start);
} // Fl_Button* start
{ quit = new Fl_Button(90, 80, 105, 30, "QUIT");
  quit->box(FL_NO_BOX);
  quit->color((Fl_Color)128);
  quit->labelsize(41);
  quit->labelcolor(FL_BACKGROUND2_COLOR);
  quit->callback((Fl_Callback*)cb_quit);
} // Fl_Button* quit
{ map = new Fl_Box(0, 0, 720, 450);
} // Fl_Box* map
{ zombie_right = new Fl_Box(-70, 193, 70, 27);
} // Fl_Box* zombie_right
{ zombie_left = new Fl_Box(805, 193, 50, 27);
} // Fl_Box* zombie_left
{ zombie_up = new Fl_Box(345, 477, 50, 71);
} // Fl_Box* zombie_up
{ zombie_down = new Fl_Box(350, -78, 50, 78);
} // Fl_Box* zombie_down
{ Ham = new Fl_Box(550, 365, 83, 64);
} // Fl_Box* Ham
{ player = new Fl_Box(331, 196, 57, 57);
  player->deactivate();
} // Fl_Box* player
{ bullet_right = new Fl_Box(360, 214, 30, 16);
  bullet_right->callback((Fl_Callback*)cb_bullet_right);
} // Fl_Box* bullet_right
{ bullet_left = new Fl_Box(330, 214, 30, 16);
  bullet_left->callback((Fl_Callback*)cb_bullet_left);
} // Fl_Box* bullet_left
{ bullet_up = new Fl_Box(345, 199, 30, 16);
  bullet_up->callback((Fl_Callback*)cb_bullet_up);
} // Fl_Box* bullet_up
{ bullet_down = new Fl_Box(345, 229, 30, 16);
} // Fl_Box* bullet_down
{ music = new SoundBox(640, 356, 25, 12);
  music->box(FL_NO_BOX);
  music->color(FL_BACKGROUND_COLOR);
  music->selection_color(FL_BACKGROUND_COLOR);
  music->labeltype(FL_NORMAL_LABEL);
  music->labelfont(0);
  music->labelsize(14);
  music->labelcolor(FL_FOREGROUND_COLOR);
  music->align(Fl_Align(FL_ALIGN_CENTER));
  music->when(FL_WHEN_RELEASE);
} // SoundBox* music
{ replay = new Fl_Button(287, 225, 140, 30, "REPLAY");
  replay->box(FL_NO_BOX);
  replay->labelsize(35);
  replay->labelcolor((Fl_Color)1);
```

```
    replay->callback((Fl_Callback*)cb_replay);
  } // Fl_Button* replay
  { credit_button = new Fl_Button(540, 145, 125, 35, "Credits");
    credit_button->box(FL_NO_BOX);
    credit_button->labelsize(41);
    credit_button->callback((Fl_Callback*)cb_credit_button);
  } // Fl_Button* credit_button
  { title = new Fl_Box(100, 287, 524, 152);
  } // Fl_Box* title
  { instructions = new Fl_Box(0, 8, 720, 450, "label");
  } // Fl_Box* instructions
  { instr_button = new Fl_Button(33, 145, 235, 36, "INSTRUCTIONS");
    instr_button->box(FL_NO_BOX);
    instr_button->labelsize(31);
    instr_button->callback((Fl_Callback*)cb_instr_button);
  } // Fl_Button* instr_button
  { Credits = new Fl_Box(0, 0, 720, 450);
  } // Fl_Box* Credits
  { menu_button = new Fl_Button(486, 0, 234, 33, "MAIN MENU");
    menu_button->box(FL_NO_BOX);
    menu_button->labelsize(41);
    menu_button->labelcolor((Fl_Color)1);
    menu_button->callback((Fl_Callback*)cb_menu_button);
  } // Fl_Button* menu_button
  { x2loud4u = new Fl_Button(520, 420, 190, 30, "Music Too Loud?");
    x2loud4u->box(FL_NO_BOX);
    x2loud4u->labelsize(24);
    x2loud4u->labelcolor((Fl_Color)1);
    x2loud4u->callback((Fl_Callback*)cb_x2loud4u);
  } // Fl_Button* x2loud4u
  { speedupbox = new Fl_Box(60, 35, 280, 45, "Zombies Are Speeding Up!!!");
    speedupbox->labelsize(22);
    speedupbox->labelcolor((Fl_Color)1);
  } // Fl_Box* speedupbox
  { snoopbox = new Fl_Box(648, 260, 56, 120);
  } // Fl_Box* snoopbox
  window->end();
} // Fl_Double_Window* window
menu_button->hide();
replay->hide();
player->hide(); //hides all sprites and images first, except for logo
map->hide();
start->hide();
quit->hide();
instr_button->hide();
instructions->hide();
GameOver->hide();
bullet_right->hide();
bullet_left->hide();
bullet_up->hide();
```

```
bullet_down->hide();
Ham->hide();
credit_button->hide();
Credits->hide();
x2loud4u->hide();
speedupbox->hide();
snoopbox->hide();
death->hide(); //hideeeeee
gameoverbackground->hide();

menumusic->play();
menumusic->setRepeat(true); //starts main menu music

//dynamically loading images

gameoverbackground->image(new Fl_PNG_Image("InstructionsBW.png"));
Ham->image(new Fl_PNG_Image("Hammy.png"));
zombie_right->image(new Fl_PNG_Image("zombie_right.png"));
zombie_left->image(new Fl_PNG_Image("zombie_left.png"));
zombie_down->image(new Fl_PNG_Image("zombie_down.png"));
zombie_up->image(new Fl_PNG_Image("zombie_up.png"));
bullet_right->image(new Fl_PNG_Image("Bullet_right.png"));
bullet_left->image(new Fl_PNG_Image("Bullet_left.png"));
bullet_up->image(new Fl_PNG_Image("Bullet_up.png"));
bullet_down->image(new Fl_PNG_Image("Bullet_down.png"));

for (int i = 0; i < LOGO_FRAMES; i++)
{
      ostringstream oss;
      oss << i;
      string fname = "logo" + oss.str() +".png";
      logo[i] = new Fl_PNG_Image ( fname.c_str());
}

for (int i = 0; i < TOP_DEATH_FRAMES; i++) //Load death animation
{
      ostringstream oss2;
      oss2 << i;
      string topdeathgo = "topham_dead" + oss2.str() +".PNG";
      topdeath[i] = new Fl_PNG_Image ( topdeathgo.c_str());
}

for (int i = 0; i < CREDITS_FRAMES; i++) //Load credits frames into an array
{
      ostringstream oss3;
      oss3 << i;
      string creditsgo = "credits" + oss3.str() +".png";
      creditsa[i] = new Fl_PNG_Image ( creditsgo.c_str());
}
```

```
  //loading zombie death animation into the arrays
  for (int i = 0; i < ZOMBIE_RIGHT_DEAD_FRAMES; i++)
  {
        ostringstream oss4;
        oss4 << i;
        string zombie_right_dead = "zombie_right_dead" + oss4.str() +".png";
        right_zombied[i] = new Fl_PNG_Image ( zombie_right_dead.c_str());
  }

  for (int i = 0; i < ZOMBIE_LEFT_DEAD_FRAMES; i++)
  {
        ostringstream oss5;
        oss5 << i;
        string zombie_left_dead = "zombie_left_dead" + oss5.str() +".png";
        left_zombied[i] = new Fl_PNG_Image ( zombie_left_dead.c_str());
  }

  for (int i = 0; i < ZOMBIE_UP_DEAD_FRAMES; i++)
  {
        ostringstream oss6;
        oss6 << i;
        string zombie_up_dead = "zombie_up_dead" + oss6.str() +".png";
        up_zombied[i] = new Fl_PNG_Image ( zombie_up_dead.c_str());
  }

  for (int i = 0; i < ZOMBIE_DOWN_DEAD_FRAMES; i++)
  {
        ostringstream oss7;
        oss7 << i;
        string zombie_down_dead = "zombie_down_dead" + oss7.str() +".png";
        down_zombied[i] = new Fl_PNG_Image ( zombie_down_dead.c_str());
  }

  for (int i = 0; i < SNOOP_FRAMES; i++)
  {
        ostringstream oss8;
        oss8 << i;
        string snoopstr = "snoop" + oss8.str() +".png";
        snoopa[i] = new Fl_PNG_Image ( snoopstr.c_str());
  }


  Fl::add_timeout(0.001, logo_animation); //starts game with logo
  Fl::add_timeout(4.5, logo_fadeout); // fades logo out 4.7
  Fl::add_timeout(8.2, mainmenu_load); //brings up the main menu screen 8.5
  Fl::add_timeout(12, item_delay); // shows buttons
  window->show(argc, argv);
  return Fl::run();
}
```

```
void back_to_menu(void*) {
  background->image(new Fl_JPEG_Image("mainmenu.jpg"));

  background->show(); //Get back to where it was before
  start->show();
  quit->show();
  title->show();
  instr_button->show();
  Ham->show();

  Ham->position(550, 365);
  quit->position(90,80); //Move quit back to where it was.

  map->hide(); //Graveyard: Exit Stage Right.
  zombie_right->hide(); //Hide zombies.
  zombie_left->hide();
  zombie_down->hide();
  zombie_up->hide();
  replay->hide(); //Hide the replay button.
  bullet_right->hide(); //Hide the bullets.
  bullet_left->hide();
  bullet_up->hide();
  bullet_down->hide();
  menu_button->hide(); //Turn off the menu_button; we don't need it on the main menu
  instructions->hide(); //We assume the player doesn't need instructions anymore.
  scorewin->hide(); //We don't need the score counter right now.
  death->hide();
  Credits->hide();
  credit_button->show();
  creditmusic->stop();
  x2loud4u->hide();
  snoopbox->hide();
  Fl::remove_timeout(snoop_animation);

  menumusic->play();

  window->redraw();
}

/**
   This happens when the credits button is clicked
*/
void display_credits(void*) {
  replay->hide();
  player->hide(); //hides all sprites and images first, except for logo
  map->hide();
  start->hide();
  quit->hide();
  instr_button->hide();
```

```
  instructions->hide();
  GameOver->hide();
  bullet_right->hide();
  bullet_left->hide();
  bullet_up->hide();
  bullet_down->hide();
  Ham->hide();
  credit_button->hide();
  death->hide(); //hideeeeee

  menu_button->hide();




  Fl::add_timeout(0.001, logo_animation); //starts credits with logo
  Fl::add_timeout(4.7, logo_fadeout); // fades logo out


  Fl::add_timeout(8, credits_animation); //fades in the credits

  Fl::add_timeout(15, snoop_animation);
}

void credits_animation(void*) {
  static int i = 0;  //fades in credits
  Credits->show();
  Credits->image(creditsa[i]);
  i++;
  Credits->parent()->redraw();
  if (i >= CREDITS_FRAMES) //pauses incriment of frame # once it reaches maximum frames
  {
  i = 0;
  Credits->image(new Fl_PNG_Image("credits.png"));
  menu_button->show();
  x2loud4u->show();
  Fl::remove_timeout(credits_animation);
  }
  else
  {
  Fl::repeat_timeout(.1, credits_animation);
  }
}

/**
   This function selects which zombie to spawn and how fast to spawn in
*/
void select(void*) {
  srand(time(0)); //seeeeeeeeed
```

```
int r = rand();//random number is generated here to select a case
int randspawn=r % 4; //four directions for zombies

double SPAWN_RATE = (rand()% 2) + Spawn_Variable;


if (upzombie == 1 and downzombie == 1 and leftzombie == 1 and rightzombie == 1)
{ //checks if all zombies are on the board
randspawn = 5;
}
else
{
      //this loop checks if the program has selected to spawn active zombie
      while ((randspawn == 0 and upzombie == 1) or
      (randspawn == 1 and downzombie == 1) or
      (randspawn == 2 and leftzombie == 1) or
      (randspawn == 3 and rightzombie == 1))

      {
              //cout << "duplicate spawn detected" << endl;
              r = rand();
              randspawn = r % 4; //retry until we haven't chosen an active zombie
      }
}
      //cout << randspawn << endl;

switch(randspawn)
{

case 0: //zombie going up
      zombie_up->position(345, 440);
      zombie_up->show(); //zombie_down->hide(); zombie_left->hide(); zombie_right->hide();
      /*Fl::remove_timeout(cb_move_down);
      Fl::remove_timeout(cb_move_left);
      Fl::remove_timeout(cb_move_right);*/
      Fl::add_timeout(0.03, cb_move_up);
      upzombie = 1;

break;

case 1: //zombie going down
      zombie_down->position(350,-78);
      zombie_down->show(); //zombie_up->hide(); zombie_left->hide(); zombie_right->hide();
      /*Fl::remove_timeout(cb_move_up);
      Fl::remove_timeout(cb_move_left);
      Fl::remove_timeout(cb_move_right);*/
      Fl::add_timeout(0.03, cb_move_down);
      downzombie = 1;

break;
```

```
  case 2: //zombie going left
        zombie_left->position(722, 188);
        zombie_left->show(); //zombie_down->hide(); zombie_right->hide(); zombie_up->hide();
        /*Fl::remove_timeout(cb_move_down);
        Fl::remove_timeout(cb_move_up);
        Fl::remove_timeout(cb_move_right);*/
        Fl::add_timeout(0.03, cb_move_left);
        leftzombie = 1;

  break;

  case 3: //zombie going right
        zombie_right->position(-70, 193);
        zombie_right->show(); //zombie_down->hide(); zombie_left->hide(); zombie_up->hide();
        /*Fl::remove_timeout(cb_move_down);
        Fl::remove_timeout(cb_move_left);
        Fl::remove_timeout(cb_move_up);*/
        Fl::add_timeout(0.03, cb_move_right);
        rightzombie = 1;

  break;

  case 5: //goes to this if all zombies are active and does nothing
  //cout << "all zombies have spawned" << endl;
  break;
  }


  Fl::repeat_timeout(SPAWN_RATE, select); // repeats function
}

void cb_move_right(void *) {
  const int SPEED = ZombieSpeed;


        if(collision(3, zombie_right->x())) //checks if player is hit
        {
                //cout << "player died" << endl;
                Fl::add_timeout(0.1, player_death);
                Fl::remove_timeout(cb_move_right);
                }
        else //keeps moving if not
        {
                zombie_right->position(zombie_right->x()+SPEED, zombie_right->y());
                Fl::repeat_timeout(SpriteFrameRate, cb_move_right);
                }

        //if (zombie_right->x() > 450 )
        //Fl::remove_timeout(cb_move_right);
```

```
  window->redraw();
}

void cb_move_up(void *) {
  const int SPEED = ZombieSpeed ;


        //if (zombie_up->y() < 0)
        //Fl::remove_timeout(cb_move_up);


        if(collision(0, zombie_up->y())) //checks if player is hit
        {
                //cout << "player died" << endl;
                Fl::add_timeout(0.1, player_death);
                Fl::remove_timeout(cb_move_up);
                }
        else //keeps moving if not
        {
                zombie_up->position(zombie_up->x(), zombie_up->y()-SPEED);
                Fl::repeat_timeout(SpriteFrameRate, cb_move_up);
                }



  window->redraw();
}

void cb_move_down(void *) {
  const int SPEED = ZombieSpeed;


        //if (zombie_down->y() > 450)
        //Fl::remove_timeout(cb_move_down);


        if(collision(1, zombie_down->y())) //checks if player is hit
        {
                //cout << "player died" << endl;
                Fl::add_timeout(0.1, player_death);
                Fl::remove_timeout(cb_move_down);
                }
        else //keeps moving if not
        {
                zombie_down->position(zombie_down->x(), zombie_down->y()+SPEED);
                Fl::repeat_timeout(SpriteFrameRate, cb_move_down);
                }


  window->redraw();
```

```
}

void cb_move_left(void *) {
  const int SPEED = ZombieSpeed ;


        //if (zombie_left->x() <0)
        //Fl::remove_timeout(cb_move_left);


        if(collision(2, zombie_left->x())) //checks if player is hit
        {
                //cout << "player died" << endl;
                Fl::add_timeout(0.1, player_death);
                Fl::remove_timeout(cb_move_left);
                }
        else //keeps moving if not
        {
                zombie_left->position(zombie_left->x()-SPEED, zombie_left->y());
                Fl::repeat_timeout(SpriteFrameRate, cb_move_left);
                }

  window->redraw();
}

void logo_animation(void*) {
  player->hide(); //hides all sprites and images first, except for logo
  map->hide();
  title->hide();
  start->hide();
  quit->hide();
  GameOver->hide();

  static int i = 0;  //fades in logo
  background->image(logo[i]);
  i++;
  background->parent()->redraw();
  if (i >= LOGO_FRAMES) //pauses incriment of frame # once it reaches maximum frames
  {
  i=0;
  background->image(new Fl_PNG_Image("logo.png"));
  Fl::remove_timeout(logo_animation);
  }
  else
  {
  Fl::repeat_timeout(.1, logo_animation);
  }
}

void logo_load() {
```

```
  //TAKE CARE OF BUTTONS
  map->hide();
  title->hide();
  start->hide();
  quit->hide();

  //LOAD LOGO INTO PLACE
  background->image(new Fl_PNG_Image("logo.png"));
  background->parent()->redraw();
}

void logo_fadeout(void*) {
  Fl::remove_timeout(logo_animation);
  static int i = LOGO_FRAMES;
  i--;
  background->image(logo[i]);
  window->redraw();

  if (i > 0)
  {
  Fl::repeat_timeout(.1,logo_fadeout);
  }
  else
  {
  i=LOGO_FRAMES;
  Fl::remove_timeout(logo_fadeout);
  }
}

void mainmenu_load(void*) {
  background->image(new Fl_JPEG_Image("mainmenu.jpg"));

  background->parent()->redraw();
  Fl::remove_timeout(logo_animation);
}

void item_delay(void*) {
  title->image(new Fl_PNG_Image("title.png"));
  start->show();
  quit->show();
  title->show();
  instr_button->show();
  title->parent()->redraw();
  background->parent()->redraw();
  Ham->show();
  credit_button->show();
}

/**
   This is called when the user presses the instructions button.
```

```
*/
void display_instr(void*) {
  instructions->image(new Fl_PNG_Image("Instructions.png"));
  instructions->show();
  background->hide();
  instr_button->hide();
  menu_button->show();
  credit_button->hide();
  window->redraw();
}

bool collision(int dir, int cor) {
  switch (dir)
  {
        case 0: //for zombie going up
        if (cor <= 210) //orignally 253
                {
                        return true;
                        }
                else
                {
                        return false;
                        }
        break;

        case 1: //for zombie going down
        if (cor >= 153) //originally 196
                {
                        return true;
                        }
                else
                {
                        return false;
                        }
        break;

        case 2: //for zombie going left

        if (cor <= 360) //originally 388
                {
                        return true;
                        }
                else
                {
                        return false;
                        }
        break;

        case 3: //for zombie going right
        if (cor >= 303) //originally 331
```

```
                {
                        return true;                    }
                else
                {
                        return false;
                        }
        break;
        }
}

void player_facing(DIRECTION d) {
  switch(d) // Load the appropriate image for where the hero is facing.
  {
        case RIGHT:
                player->image(new Fl_PNG_Image("topham_right.png"));
                player->parent()->redraw();
                break; //right
        case DOWN:
                player->image(new Fl_PNG_Image("topham_down.png"));
                player->parent()->redraw();
                break; //down
        case LEFT:
                player->image(new Fl_PNG_Image("topham_left.png"));
                player->parent()->redraw();
                break; //left
        case UP:
                player->image(new Fl_PNG_Image("topham_up.png"));
                player->parent()->redraw();
                break; //up
  }
}

void set_gamescreen(int startcheck, int replaycheck) {
  gameoncheck = 1; //saying that the game is on

  Spawn_Variable = 1;
  zombiecount = 0; /* The count of the zombies need to be reset too, such that
  the counter will be accurate if the player hits "replay". */

  scoreout->hide(); // Now we restart the score counter.
  scoreout->value("0");
  scoreout->redraw();
  scoreout->show();

  ZombieSpeed = 2; /* ...Along with the zombie speed. */



  //Shows player and game environment; hides main menu and buttons
  credit_button->hide();
```

```
  Ham->show();
  Ham->position(331, 196);
  player->show();
  title->hide();
  start->hide();
  instr_button->hide();
  quit->hide();
  map->show();
  death->hide();
  gameoverbackground->hide();
  map->image(new Fl_PNG_Image("map.png"));
  background->parent()->redraw();
  Fl::add_timeout(1.0, select);

  if(startcheck)
  { // If this function is called by the start button...
  menumusic->stop(); // Stop the main menu music and...
  scorewin->show(); // Show a window containing the player's score.
  } //We don't need to do this for the replay button, so startcheck ensures we don't.

  keytotal = 40000;
  player->image(new Fl_PNG_Image("topham_right.png"));

  if(replaycheck)
  { // If the function is called by the replay button...
  replay->hide(); // Hide the replay button and...
  menu_button->hide(); // Hide the main menu button.

  //replaymusic->play(); // Play the death music backwards. (moved to callback in replay buttc
  }//Like stopping menumusic, we don't need to hide replay with the start button.

  gamemusic->play(); //start game music
  gamemusic->setRepeat(true);

  bullet_reset(0); //resetting bullets to fix bug
  bullet_reset(1); //bug: could not shoot if bullet was mid-flight and player died
  bullet_reset(2);
  bullet_reset(3);
}

/**
   this moves the bullet see comments within, the other three move functions are basically clo
*/
void bullet_move_right(void *) {
  const int SPEED = BulletSpeed;

  if (bullet_right->x()>=720){ // if bullet goes off-screen
        bullet_reset(0);} // reset bullet to original location
  else
  {
```

```
        bullet_right->show(); // makes bullet appear when shooting
        bullet_right->position(bullet_right->x()+SPEED, bullet_right->y()); // moves bullet
        Fl::repeat_timeout(SpriteFrameRate, bullet_move_right); // repeats function bullet_move_ri

        if(bullet_collision(0)) // checks if bullet collided with zambie
        {zombie_death(0,zombiecount); // does stuff that should happen when zambie dies
        bullet_reset(0);} // resets bullet
  }
}

void bullet_move_up(void *) {
  const int SPEED = BulletSpeed;

  if (bullet_up->y()<=0){
        bullet_reset(3);}
  else
  {
        bullet_up->show();
        bullet_up->position(bullet_up->x(), bullet_up->y()-SPEED);
        Fl::repeat_timeout(SpriteFrameRate, bullet_move_up);

        if(bullet_collision(3))
        {zombie_death(3,zombiecount);
        bullet_reset(3);}
  }
}

void bullet_move_down(void *) {
  const int SPEED = BulletSpeed;

  if (bullet_down->y()>=420){
        bullet_reset(1);}
  else
  {
        bullet_down->show();
        //bullet_down->image(new Fl_PNG_Image("Bullet_down.png"));
        bullet_down->position(bullet_down->x(), bullet_down->y()+SPEED);
        Fl::repeat_timeout(SpriteFrameRate, bullet_move_down);

        if(bullet_collision(1))
        {zombie_death(1,zombiecount);
        bullet_reset(1);}

  }
}

void bullet_move_left(void *) {
  const int SPEED = BulletSpeed;

  if (bullet_left->x()<=0){
```

```
        bullet_reset(2);}
  else
  {
        bullet_left->show();
        bullet_left->position(bullet_left->x()-SPEED, bullet_left->y());
        Fl::repeat_timeout(SpriteFrameRate, bullet_move_left);

        if(bullet_collision(2))
        {zombie_death(2,zombiecount);
        bullet_reset(2);}

  }
}

void right_zombie_death_animation(void*) {
  static int i = 1;  // Begin death animation.
  zombie_right->image(right_zombied[i]);
  i++;
  zombie_right->parent()->redraw();
  if (i >= ZOMBIE_RIGHT_DEAD_FRAMES) //pauses increment of frame # once it reaches maximum frames
  {
  i = 1;
  rightzombie = 0;
  zombie_right->hide();
  zombie_right->position(-70,193); //move zombie back
  Fl::remove_timeout(right_zombie_death_animation);
  zombie_right->image(new Fl_PNG_Image("zombie_right.png"));
  }
  else
  {
  Fl::repeat_timeout(.15, right_zombie_death_animation);
  }
}

void left_zombie_death_animation(void*) {
  static int i = 1;  // Begin death animation.
  zombie_left->image(left_zombied[i]);
  zombie_left->parent()->redraw();
  i++;
  if (i >= ZOMBIE_LEFT_DEAD_FRAMES) //pauses increment of frame # once it reaches maximum frames
  {
  i = 1;
  zombie_left->hide(); //hides zombie
  zombie_left->position(805,193); //move zombie back
  leftzombie = 0;
  Fl::remove_timeout(left_zombie_death_animation);
  zombie_left->image(new Fl_PNG_Image("zombie_left.png"));
  }
  else
  {
```

```
  Fl::repeat_timeout(.15, left_zombie_death_animation);
  }
}

void up_zombie_death_animation(void*) {
  static int i = 1;  // Begin death animation.
  zombie_up->image(up_zombied[i]);
  zombie_up->parent()->redraw();
  i++;
  if (i >= ZOMBIE_UP_DEAD_FRAMES) //pauses increment of frame # once it reaches maximum frames
  {
  i = 1;
  zombie_up->hide();
  zombie_up->position(345,477); //move zombie back
  upzombie = 0;
  Fl::remove_timeout(up_zombie_death_animation);
  zombie_up->image(new Fl_PNG_Image("zombie_up.png"));
  }
  else
  {
  Fl::repeat_timeout(.15, up_zombie_death_animation);
  }
}

void down_zombie_death_animation(void*) {
  static int i = 1;  // Begin death animation.
  zombie_down->image(down_zombied[i]);
  zombie_down->parent()->redraw();
  i++;
  if (i >= ZOMBIE_DOWN_DEAD_FRAMES) //pauses increment of frame # once it reaches maximum frames
  {
  i = 1;
  downzombie = 0;
  zombie_down->hide();
  zombie_down->position(345,-78);
  Fl::remove_timeout(down_zombie_death_animation);
  zombie_down->image(new Fl_PNG_Image("zombie_down.png"));
  }
  else
  {
  Fl::repeat_timeout(.15, down_zombie_death_animation);
  }
}

/**
   detects collision between bullet and zombie
*/
bool bullet_collision(int dir) {
  switch (dir)
  {
```

```
        case 3: //for bullet going up
        if (bullet_up->y()<=zombie_down->y()+20 //If the bullet hits the zombie
                and zombie_down->y() > 0) //in the window...
                {
                        return true; //it hits!
                        }
                else
                {
                        return false; //no dice
                        }
        break;

        case 1: //for bullet going down
        if (bullet_down->y() >= zombie_up->y()+20       //If the bullet hits the zombie
                and zombie_up->y() + zombie_up->h() > 0) //in the window...
                {
                        return true; //it hits!
                        }
                else
                {
                        return false; //no dice
                        }
        break;

        case 2: //for bullet going left

        if (bullet_left->x() <= zombie_right->x()+20 //If the bullet hits the zombie
                and zombie_right->x() + zombie_right->w() < window->w() + window->x()) //in the
                {
                        return true;
                        }
                else
                {
                        return false;
                        }
        break;

        case 0: //for bullet going right
        if (bullet_right->x() >= zombie_left->x()+20 //If the bullet hits the zombie
                and zombie_left->x() < window->x() + window->w()) //in the window...
                {
                        return true; //it hits!
                        }
                else
                {
                        return false; //no dice.
                        }
        break;
        }
}
```

```
/**
   Called when zombie dies.
*/
void zombie_death(int dir, double& counter) {
  ostringstream oss;

  switch(dir)
  {
        case 0: // right bullet hitting zombie left, hides zambie
        Fl::remove_timeout(cb_move_left); //stops movement of zombie
        Fl::add_timeout(0.001, left_zombie_death_animation);
        ++counter; //Now we iterate the zombie count...
        //cout << "counter: " << counter << endl;

        adjust_difficulty(counter,ZombieSpeed); //AHAHAHAHAAAA! (then we check difficulty)

        oss << counter; //Store the counter in an outputstringstream object
        scoreout->hide();
        scoreout->value(oss.str().c_str()); //...and display it in scoreout.
        scoreout->redraw(); //redraw; otherwise, it looks stupid.
        scoreout->show(); //Show our score.
        break;

        case 1: // same for down zamb

        Fl::remove_timeout(cb_move_up);
        Fl::add_timeout(0.001, up_zombie_death_animation);

        ++counter; //Now we iterate the zombie count...
        //cout << "counter: " << counter << endl;

        adjust_difficulty(counter,ZombieSpeed); //AHAHAHAHAAAA! (then we check difficulty)

        oss << counter; //Store the counter in an outputstringstream object
        scoreout->hide();
        scoreout->value(oss.str().c_str()); //...and display it in scoreout.
        scoreout->redraw(); //redraw; otherwise, it looks stupid.
        scoreout->show(); //Show our score.
        break;

        case 2: // same for zamb goin left

        Fl::remove_timeout(cb_move_right);
        Fl::add_timeout(0.001, right_zombie_death_animation);

        ++counter; //Now we iterate the zombie count...
        //cout << "counter: " << counter << endl;

        adjust_difficulty(counter,ZombieSpeed); //AHAHAHAHAAAA! (then we check difficulty)
```

```
        oss << counter; //Store the counter in an outputstringstream object
        scoreout->hide();
        scoreout->value(oss.str().c_str()); //...and display it in scoreout.
        scoreout->redraw(); //redraw; otherwise, it looks stupid.
        scoreout->show(); //Show our score.
        break;

        case 3: // up yo

        Fl::remove_timeout(cb_move_down);
        Fl::add_timeout(0.001, down_zombie_death_animation);

        ++counter; //Now we iterate the zombie count...
        //cout << "counter: " << counter << endl;

        adjust_difficulty(counter,ZombieSpeed); //AHAHAHAHAAAA! (then we check difficulty)

        oss << counter; //Store the counter in an outputstringstream object
        scoreout->hide();
        scoreout->value(oss.str().c_str()); //...and display it in scoreout.
        scoreout->redraw(); //redraw; otherwise, it looks stupid.
        scoreout->show(); //Show our score.
        break;

  }
}

void adjust_difficulty(int counternum,int& movingspeed) {
  if(counternum % 10 == 0 and counternum > 0)
  { // If the player's score is a multiple of 10...
        ++movingspeed; // ...make the zombies go faster.
        Fl::add_timeout(0.001, show_speedupmsg);
        speedupsound->stop();
        speedupsound->play();
        //cout << "Speed up! Zombies going " << movingspeed << " pixels now! \n";
  } // Initially, we will try a linear difficulty curve.

  if(counternum % 5 == 0 and counternum > 0 and Spawn_Variable > 0.1)
  { //if the player's score is a multiple of 5...
        Spawn_Variable -= 0.07; //...make the time between the zombies spawning shorter.
        //cout << "Spawn rate increased!" << endl;
  } //this is an added factore fo difficulty.
}

void show_speedupmsg(void*) {
  speedupbox->show();

  static int i = 0;
  if (i<7)
```

```
        {
        i++;
        if (i%2 == 0)
                {
                speedupbox->position(65, 35);
                }
                else
                {
                speedupbox->position(380, 35);
                }
        Fl::repeat_timeout(0.3, show_speedupmsg);
        }
        else
        {
        i=0;
        speedupbox->hide();
        Fl::remove_timeout(show_speedupmsg);
        }
}

/**
   this function is called whenever a bullet needs to be reset
*/
void bullet_reset(int dir) {
  switch(dir)
  {
        case 0: // bullet going right, BRINE

        bullet_right->position(360,214); //resets bullet to original position
        Fl::remove_timeout(bullet_move_right); //stops bullet movement
        bullet_right->hide(); //hides bullet

        break;

        case 1: // bullet goin down bruh

        bullet_down->position(345,229);
        Fl::remove_timeout(bullet_move_down);
        bullet_down->hide();

        break;

        case 2: // goin LEFT NOW

        bullet_left->position(330,214);
        Fl::remove_timeout(bullet_move_left);
        bullet_left->hide();

        break;
```

```
        case 3: // WE UP NAO

        bullet_up->position(345,199);
        Fl::remove_timeout(bullet_move_up);
        bullet_up->hide();

        break;
  }
}

void gunfire(int orient) {
  if (gameoncheck == 1) //if the game is on
  {
        gunshot->play(); // Play a gunshot noise.
        switch(orient)
        {
                case 0: //cout << "Bullet goes RIGHT." << endl;
                Fl::add_timeout(0.01, bullet_move_right); //calls the function that fires the bullet

                        break;

                case 1: //cout << "Bullet goes DOWN." << endl;
                Fl::add_timeout(0.01, bullet_move_down);

                        break;

                case 2: //cout << "Bullet goes LEFT." << endl;
                Fl::add_timeout(0.01, bullet_move_left);


                        break;

                case 3: //cout << "Bullet goes UP." << endl;
                Fl::add_timeout(0.01, bullet_move_up);

                        break; //#linkinparksucks //yup //#personallyipreferthepair
        }
  }
}

void player_death(void*) {
  gameoncheck = 0; //saying game is off

  //cout << "Initiate 'player_death'. \n \n";

  player->hide(); //hide instances
  background->hide();
  map->hide();

  zombie_right->hide();
```

```
  zombie_left->hide();
  zombie_down->hide();
  zombie_up->hide();
  speedupbox->hide();

  speedupsound->stop();

  Fl::remove_timeout(show_speedupmsg);
  Fl::remove_timeout(select); //end timeouts
  Fl::remove_timeout(cb_move_down);
  Fl::remove_timeout(cb_move_left);
  Fl::remove_timeout(cb_move_right);
  Fl::remove_timeout(cb_move_up);

  leftzombie = 0; //these reset the algorithm for detecting if a zombie has spawned or not
  rightzombie = 0;
  upzombie = 0;
  downzombie = 0;

  gamemusic->stop();
  deathmusic->play();

  death->show();


  Fl::add_timeout(0.001, top_death_animation); //starts death cutscene

  window->redraw();
}

void top_death_anim_reversal(void*) {
  Fl::remove_timeout(top_death_animation);
  static int i = TOP_DEATH_FRAMES;
  i--;
  background->image(topdeath[i]);
  background->parent()->redraw();
  if (i >= 2)
  {
  Fl::repeat_timeout(.1, top_death_anim_reversal);
  }
}

void top_death_load() {
  //TAKE CARE OF BUTTONS
  quit->hide();

  map->hide(); //Graveyard: Exit Stage Right.
  zombie_right->hide(); //Hide zombies.
  zombie_left->hide();
  zombie_down->hide();
```

```
  zombie_up->hide();
  replay->hide(); //Hide the replay button.
  bullet_right->hide(); //Hide the bullets.
  bullet_left->hide();
  bullet_up->hide();
  bullet_down->hide();
  menu_button->hide(); //Turn off the menu_button; we don't need it on the main menu
  instructions->hide(); //We assume the player doesn't need instructions anymore.

  //LOAD DEATH ANIM. INTO PLACE
  background->image(new Fl_PNG_Image("topham_dead13.PNG"));
  window->redraw();
}

void top_death_animation(void*) {
  player->hide(); //hides all sprites and images first, except for logo
  map->hide();
  title->hide();
  start->hide();
  quit->hide();
  GameOver->hide();
  Ham->hide();

  static int i = 0;  // Begin death animation.
  death->image(topdeath[i]);
  i++;

  background->parent()->redraw();
  if (i >= TOP_DEATH_FRAMES) //pauses increment of frame # once it reaches maximum frames
  {
  i = 0;
  background->image(new Fl_PNG_Image("topham_dead13.png"));
  Fl::remove_timeout(top_death_animation);
  Fl::add_timeout(1.0, call_gameover);
  }
  else
  {
  Fl::repeat_timeout(.1, top_death_animation);
  }
}

void top_death_fadeout(void*) {
  GameOver->hide();
  replay->hide();
  quit->hide();
  menu_button->hide();

  death->show();
  static int i = TOP_DEATH_FRAMES;
```

```
  i--;
  death->image(topdeath[i]);
  death->parent()->redraw();
  if (i >= 2)
  {
  Fl::repeat_timeout(0.1, top_death_fadeout);
  }
  else
  {
  i = TOP_DEATH_FRAMES;
  set_gamescreen(0,1); //Set up the main game.
  Fl::remove_timeout(top_death_fadeout);
  }
}

void call_gameover(void*) {
  death->hide();
  gameoverbackground->show();
  gameoverbackground->parent()->redraw();


  GameOver->show();
  replay->show();
  //Move the quit button and give the player the option to quit.
  quit->position(300,280);
  quit->show();
  menu_button->show();
}

void snoop_animation(void*) {
  snoopbox->show();

  static int i = 1;  //fades in logo
  snoopbox->image(snoopa[i]);
  i++;
  snoopbox->parent()->redraw();
  if (i >= SNOOP_FRAMES) //pauses incriment of frame # once it reaches maximum frames
  {
  i=1;
  //snoopbox->image(new Fl_PNG_Image("snoop1.png"));
  }

  Fl::repeat_timeout(.03, snoop_animation);
}
/**
```

5 Test

5.1 Expected Results

We expect to see a fully-functional game.

5.2 Actual Results


```
*/
//
/**
```

6 Development Change Log

When making a comment here on bug fixes or additions to the game, please describe what you did, what functions were used, and initial your comment with a date and version number.

6.1 Additions Made:

Replaced title from text box with an image, images is dynamically loaded under item_delay function

- NC 11/20/13 ALPHA_02

Added a box for instructions, loaded instructions image dynamically in main.
Created a function to show instructions when the instructions button is clicked.
Added a button to return to main menu and a function called ‘‘back_to_menu’’ that can be reuse
in other buttons to return to the main menu.
- NC 11/20/201 3 ALPHA_03

Provided a secondary window named scorewin that will keep track of the player’s score.
Wrote out a more detailed design section for documentation.
Commented out some less necessary functional debugging ‘‘cout’’ statements.
Added gunshot sound effects.
- BAM 11/21/2013 ALPHA_04

Added a death cutscene that plays after the player dies.
- WAT 11/21/2013 ALPHA_05

Added a short sound clip that plays when the player dies and revives.
Created a function (adjust difficulty) that increments the zombies’ speed every 10 kills.
- BAM 11/26/2013 ALPHA_06

6.2 Bug Fixes:

Fixed bug that did not reset the player’s position after pressing replay button.
Reset the variable keytotal to 40000 in the set_gamescreen function.
- NC 11/20/2013 ALPHA_02

Set bullet speed, zombie speed, and sprite frame rates to global variables for easy change.
Also set values to run game much smoother.
- NC 11/20/2013 ALPHA_03

Told the main menu button to go away once the replay button was pressed.
Removed unnecessary whitespace from the bullet sprites.
- BAM 11/21/2013 ALPHA_04

Fixed previously known bugs, including:

A redraw error within the scoring window that was preventing the readability of the score
The window now displays the score correctly without needing the user to focus on scorewin
Bullet collisions with the zombie are fixed.

- WAT 11/21/2013 ALPHA_05

Fixed a problem with the death cutscene where the buttons to replay and quit would not display
Made the zombies slow down to initial speed after the player died for fairness’s sake.
- BAM 11/26/2013 ALPHA_06

Shifted the changelog into a form easily recognized by LaTeX.
- BAM 11/26/2103 ALPHA_06.1

"Portablized" the game by editing fltk-config, making it playable on normal Windows machines.
- BAM 12/3/2013 RELEASE_02

6.3 Known Bugs (by version):

Although score was implemented, the current method to display the score is... not perfect.
Despite telling the window to redraw itself after outputting the score, numbers are overlaying
The score does not appear to display until the player focuses on the window, which is non-intu
We need to set the bullet to only collide when the zombies are in the screen!
- BAM 11/21/2013 ALPHA_04

The program takes up quite a bit of resources. In the future we may have to fix that.
Lag.
- WAT 11/21/2013 ALPHA_05

The death animation appears to be quite obstrusive; we may want to fix that.
This changelog is incompatible with LaTeX, meaning we cannot produce documentation.
The program crashes if the player replays and dies again!

    This problem is due to how the death animation has been implemented.
    The current version has the animation commented out, and it works fine.

- BAM 11/26/2013 ALPHA_06

We probably want to figure out how we can load the bullets dynamically so the documentation
is neat.
- BAM 11/26/2013 ALPHA_06.1

The game is not portable to Windows machines outside of MinGW. This is a big problem.
- BAM 11/27/2013 BETA_01.0

*/
//