Specification:

This is our final group project designed to demonstrate the various techniques we have learned in this course. We are specified to use a certain number of functions, as well as a minimum number of windows, various arrays, loops and images. The main requirement of the project was to collaborate as a team to test what it would be like to write a program in a professional environment. We will be using FLTK and Fluid for the graphical user interface, C++ for the body of the code, and LaTeX to document our project. Our team has decided to design and implement a "dancing" game.

Due to the time constraints of this assignment, our program, although functional, is not to the point which we would like it to be. If there were more time to add features, our team would like to have been able to include a score saving feature, as well as additional images in the start screen, help and Game Over windows. We also had intended on including extra levels with different backgrounds, a "Lives" feature, and an increasing difficulty as time in the game progressed.

Analysis:

- Inputs: The inputs for this project are user triggered through button presses on the keyboard. Arrow keys, as well as mouse clicks and the space bar are all required user inputs.
- Process:
  - Decide on the over all look and layout of the final design.
  - Use the internet to find various gif, jpeg and png images to imbed in the program.
  - Write psuedo code to determine the necessary functions, loops, et cetera.
  - Collaborate as a team to code the images and various inputs in to the final program.
- Outputs: The outputs for our program are the gif images which cycle regularly, as well as the pop up windows which provide the user with game play information.

Design

1. We began by brainstorming ideas for our final design. The concept changed forms a number of times, and we eventually settled on a "Dance Dance Revolution" style matching game.

2. Next we began to write a mock up of our program using psuedo code to determine what inputs were necessary and which loops, functions, et cetera, would need to be implemented.

3. We then took to the internet to find the various gif, jpeg and png images to imbed in to the program.

4. Next we initiated the implementation of our program using FLTK to design the layout of the final design.

5. The program is designed to receive inputs from the user and recognize when the inputs match what has been displayed.

6. The program utilizes a random image generator to cycle through images of a keyboard's arrow keys.

7. When a correct (matching) key is pressed within a specified time limit, the gif image on screen cycles and a new randomly generated arrow key appears on screen.

8. If the user presses the wrong key, or allows the timer to run out on the currently displayed key, the game ends and the player has lost.

9. When the player has been eliminated, a "Game Over" prompt appears on screen and the program restarts.

Function Definitions

```
main()
inputs: none
process: Called when the program is run; executes make_window, show,
loadImages, and loadLevel.
outputs: none

make_window()
imputs: none
process: FLTK-generated function, creates the window for our program
outputs: pointer to Fl_Double_Window (the window it made)

handle()
inputs: The code for the event that is being handled
process: If the event was a keypress, pass it into determineKeyPressed.
outputs: Used by FLTK only

loadImages()
inputs: none
process: Calls loadBackgrounds, loadArrowKeys, and loadGifs.
outputs: none
```

```
loadBackgrounds()
inputs: none
process: Sets background Fl image variables to images on the local
machine.
outputs: none

loadArrowKeys()
inputs: none
process: Sets arrow key Fl image variables to images on the local
machine.
outputs: none

loadGifs()
inputs: none
process: Sets dancer GIF Fl image variables to images on the local
machine.
outputs: none

loadLevel(int)
inputs: The number of the level to load (starting at 0)
process: Changes the background and dancer images to those corresponding
to the level we are loading.
outputs: none
```

```
setNewTimer()
inputs: none
process: Calls popupRandomArrow, and creates a new timeout for
timerExpire.
outputs: none

timerExpire()
inputs: none
process: Calls correctKeyPressed to see if the player had pressed the
right key; if not then end the game, if so then call setNewTimer.
outputs: none

correctKeyPressed()
inputs: none
process: Checks bCorrectKeyPressed and returns it.
outputs: returns true if the player had pressed the key within the
specified amount of time.

determineKeyPressed(int)
inputs: The code of the key that was pressed
process: Takes the input code and compares it to the code of the key
that should have been pressed; if they match, set a global bool
bCorrectKeyPressed to true.
```

```
popupRandomArrow()
inputs: none
process: Randomly picks an arrow key and displays it on the screen.
outputs: none
```

Implementation:

Here is the code for our game. We begin by include all of the necessary header files and intiating our variables.

C++ source code written to file dance.cpp

```
#include <iostream>
#include <sstream>
#include <stdlib.h>
#include <time.h>
#include "dance.h"

Game_Window* game_win;
Fl_Double_Window* menu_win;
Fl_Double_Window* help_win;
Fl_Double_Window* gameover_win;
```

This section of code creates an array of images we will use for our backgrounds.

C++ source code appended to file dance.cpp

```
Fl_JPEG_Image* i_background [2];
Fl_PNG_Image* i_presstostart;
Fl_PNG_Image* i_key_bg;

```

Here is where we create our images for the arrow keys.

C++ source code appended to file dance.cpp

```

Fl_PNG_Image* i_key_up;
Fl_PNG_Image* i_key_down;
Fl_PNG_Image* i_key_left;
Fl_PNG_Image* i_key_right;

```

This section of code is where we do the calculation to decide wether the correct key has been pressed.

C++ source code appended to file dance.cpp

```
|
|bool correctKeyPressed = false;
|bool incorrectKeyPressed = false;
|int expectedKey = 0;
|
```

When any event happens, this is called. "e" is the code for the type of event.

C++ source code appended to file dance.cpp

```
|
|int Game_Window::handle(int e)
|{
|       if(e == 12)
|       {
|               if(Fl::event_key() == expectedKey)
|                       correctKeyPressed = true;
|
|               else if(Fl::event_key() == 32) // Key for spacebar
```

C++ source code appended to file dance.cpp

```
                startPlaying();
            else
            {
                incorrectKeyPressed = true;
                correctKeyPressed = false;
            }
            std::cout << Fl::event_key() << std::endl;
        }
    return 0;
}
```

Here is the function to load the background images.

C++ source code appended to file dance.cpp

```
void loadBackgrounds()
{
        i_background[0] = new Fl_JPEG_Image("res/bg/bg_1.jpg");
        i_background[1] = new Fl_JPEG_Image("res/bg/bg_2.jpg");

        i_presstostart = new Fl_PNG_Image("res/bg/spacebar.png");
        i_key_bg = new Fl_PNG_Image("res/bg/bg_key.png");
}
```

And the function to load the arrow keys.

C++ source code appended to file dance.cpp

```
void loadArrowKeys()
{
        i_key_up = new Fl_PNG_Image("res/keys/up_key.png");
        i_key_down = new Fl_PNG_Image("res/keys/down_key.png");
        i_key_left = new Fl_PNG_Image("res/keys/left_key.png");
        i_key_right = new Fl_PNG_Image("res/keys/right_key.png");
}
```

Here, we initiate the arrays for oading the various frames of the GIF images.

C++ source code appended to file dance.cpp

```
const int carlton_N = 24;
const int justin_N = 25;
const int mj_N = 9;
const int snoop_N = 19;
Fl_GIF_Image* carlton_images[carlton_N]; // frames are 0-23
Fl_GIF_Image* justin_images[justin_N]; // frames are 0-24
Fl_GIF_Image* mj_images[mj_N]; // frames are 0-8
Fl_GIF_Image* snoop_images[snoop_N]; // frames are 0-18
```

These functions are responsible for the actual loading and cycling of the GIF images.

C++ source code appended to file dance.cpp

```
|
|void loadGifs()
|{
|    for(int i = 0; i < carlton_N; i++)
|    {
|      std::ostringstream oss;
|      oss << "res/dancers/carlton/carlton" << i << ".gif";
|      carlton_images[i] = new Fl_GIF_Image(oss.str().c_str());
|    }
|
|    for(int i = 0; i < justin_N; i++)
|    {
|      std::ostringstream oss;
|      oss << "res/dancers/justin/justin" << i << ".gif";
|      justin_images[i] = new Fl_GIF_Image(oss.str().c_str());
|    }
```

C++ source code appended to file dance.cpp

```
    for(int i = 0; i < mj_N; i++)
    {
      std::ostringstream oss;
      oss << "res/dancers/michael/mj" << i << ".gif";
      mj_images[i] = new Fl_GIF_Image(oss.str().c_str());
    }

    for(int i = 0; i < snoop_N; i++)
    {
      std::ostringstream oss;
      oss << "res/dancers/snoop/snoop" << i << ".gif";
      snoop_images[i] = new Fl_GIF_Image(oss.str().c_str());
    }

}
```

C++ source code appended to file dance.cpp

```
void animate_carlton(void*)
{
    static int i = 0;
    game_win->carlton->image(carlton_images[i]);
    game_win->carlton->parent()->redraw();
    i = (i + 1) % carlton_N;
    Fl::repeat_timeout(.075,animate_carlton);
}
void animate_justin(void*)
{
    static int i = 0;
    game_win->carlton->hide();
    game_win->justin->image(justin_images[i]);
    game_win->justin->parent()->redraw();
    i = (i + 1) % justin_N;
    Fl::repeat_timeout(.075,animate_justin);
}
```

C++ source code appended to file dance.cpp

```
void animate_mj(void*)
{
    static int i = 0;
    game_win->justin->hide();
    game_win->mj->image(mj_images[i]);
    game_win->mj->parent()->redraw();
    i = (i + 1) % mj_N;
    Fl::repeat_timeout(.075,animate_mj);
}
void animate_snoop(void*)
{
    static int i = 0;
    game_win->mj->hide();
    game_win->snoop->image(snoop_images[i]);
    game_win->snoop->parent()->redraw();
    i = (i + 1) % snoop_N;
    Fl::repeat_timeout(.075,animate_snoop);
}
```

C++ source code appended to file dance.cpp

```
void loadImages()
{
        loadBackgrounds();
        loadArrowKeys();
        loadGifs();
}
```

This part of the code loads a new lever when the player selects the start button from the GUI.

C++ source code appended to file dance.cpp

```
void loadLevel(int levelNum)
{
        game_win->box_background->image(i_background[levelNum]);

}
```

This section is where the arrow keys are randomly generated. The code is designed in such a way that the same arrow key will never present itself more than once in a row.

C++ source code appended to file dance.cpp

```
|
|void popupRandomArrow()
|{
|       static int lastNum = 0;
|
|       int randNum = randNum = rand() % 4 + 1;
|
|       while(randNum == lastNum)
|               randNum = rand() % 4 + 1;
|       lastNum = randNum;
|       Fl_PNG_Image* newArrow;
```

C++ source code appended to file dance.cpp

```
        if(randNum == 1)
        {
                newArrow = i_key_up;
                expectedKey = 65362;
        }
        else if(randNum == 2)
        {
                newArrow = i_key_down;
                expectedKey = 65364;
        }
        else if(randNum == 3)
        {
                newArrow = i_key_left;
                expectedKey = 65361;
        }
```

C++ source code appended to file dance.cpp

```
        else
        {
                newArrow = i_key_right;
                expectedKey = 65363;
        }


        game_win->box_key->show();
        game_win->box_key->image(newArrow);
        game_win->redraw();
}
```

The following code is what actually interacts with the game player. It contains the timer which will default to a "Game Over" screen when it has expired, as well as the functions to detect if the correct key has been pressed, if the incorrect key has been pressed, of if no keys were pressed.

C++ source code appended to file dance.cpp

```
|
|void timerExpire(void*);
|
|void setNewTimer()
|{
|       correctKeyPressed = false;
|       popupRandomArrow();
|       Fl::remove_timeout(timerExpire);
|       Fl::add_timeout(1.0, timerExpire);
|}
|
|void timerExpire(void*)
|{
|       std::cout << "Correct Key Pressed: " << correctKeyPressed << std::endl;
|           std::cout << "Incorrect Key Pressed: " << incorrectKeyPressed << std
```

C++ source code appended to file dance.cpp

```
        if(incorrectKeyPressed)
        {
                gameOver();
        }

        if(correctKeyPressed)
        {
                setNewTimer();
                //Fl::repeat_timeout(1.0, timerExpire);
        }
        else
        {
                gameOver();
        }

}
```

Here we set up all of the functions necessary to actually play the game.

C++ source code appended to file dance.cpp

```
void initRandomSeed()
{
        srand(time(NULL));
}

void beginKeypressSequence(void*)
{
        setNewTimer();
        //Fl::add_timeout(1.0, timerExpire);
}

void loadGameWindow()
{
        loadImages();
        loadLevel(0);
```

C++ source code appended to file dance.cpp

```
        game_win->box_presstostart->image(i_presstostart);

        game_win->show();

}
```

This is where the spacebar is set to start the game when pressed.

C++ source code appended to file dance.cpp

```
void startPlaying()
{
        Fl::add_timeout(0.2, beginKeypressSequence);
        game_win->box_presstostart->image(i_key_bg);

        Fl::add_timeout(0,animate_carlton);
        Fl::add_timeout(10,animate_justin);
        Fl::add_timeout(20,animate_mj);
        Fl::add_timeout(30,animate_snoop);
}
```

This function removes everything and kills the program once the player has lost the game.

C++ source code appended to file dance.cpp

```
|
|void gameOver()
|{
|       /*Fl::remove_timeout(animate_carlton);
|       Fl::remove_timeout(animate_carlton);
|       Fl::remove_timeout(animate_carlton);
|       Fl::remove_timeout(animate_carlton);*/
|
|       game_win->box_presstostart->image(i_presstostart);
|       game_win->box_key->hide();
|
|       game_win->hide();
|       gameover_win->show();
|}
|
```

And finally, the main function to put it all together and make it a useable program!

C++ source code appended to file dance.cpp

```
int main()
{
        game_win = make_game_window();

        menu_win = make_menu_window();
        menu_win->show();

        help_win = make_help_window();
        gameover_win = make_gameover_window();

        initRandomSeed();

        Fl::run();
}
```

GUI Code:

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

#include "dance.h"
#include <FL/Fl_Double_Window.H>

void Game_Window::cb_Quit_i(Fl_Button*, void*) {
  Main->show();
//Game_Window::hide();
}
void Game_Window::cb_Quit(Fl_Button* o, void* v) {
  ((Game_Window*)(o->parent()))->cb_Quit_i(o,v);
}
Game_Window::Game_Window(int X, int Y, int W, int H, const char *L
  : Fl_Double_Window(X, Y, W, H, L) {
  _Game_Window();
}

Game_Window::Game_Window(int W, int H, const char *L)
  : Fl_Double_Window(0, 0, W, H, L) {
  clear_flag(16);
  _Game_Window();
}
```

```
void Game_Window::_Game_Window() {
this->box(FL_FLAT_BOX);
this->color((Fl_Color)FL_GRAY0);
this->selection_color((Fl_Color)FL_BACKGROUND_COLOR);
this->labeltype(FL_NO_LABEL);
this->labelfont(0);
this->labelsize(14);
this->labelcolor((Fl_Color)FL_FOREGROUND_COLOR);
this->align(FL_ALIGN_TOP);
this->when(FL_WHEN_RELEASE);
{ box_background = new Fl_Box(0, 0, 500, 400);
  box_background->color((Fl_Color)48);
} // Fl_Box* box_background
{ box_presstostart = new Fl_Box(0, 400, 500, 100);
} // Fl_Box* box_presstostart
{ box_key = new Fl_Box(210, 410, 80, 80);
} // Fl_Box* box_key
{ carlton = new Fl_Box(170, 47, 175, 353);
} // Fl_Box* carlton
{ justin = new Fl_Box(166, 54, 135, 301);
} // Fl_Box* justin
{ mj = new Fl_Box(110, 58, 240, 317);
} // Fl_Box* mj
```

```
{ snoop = new Fl_Box(160, 88, 150, 332);
} // Fl_Box* snoop
{ Fl_Button* o = new Fl_Button(425, 5, 70, 20, "Quit");
  o->down_box(FL_DOWN_BOX);
  o->color((Fl_Color)36);
  o->selection_color((Fl_Color)43);
  o->labelcolor((Fl_Color)26);
  o->callback((Fl_Callback*)cb_Quit);
} // Fl_Button* o
end();
}

Fl_Double_Window *Help=(Fl_Double_Window *)0;

static void cb_Got(Fl_Button*, void*) {
  Main->position(200,200);
Main->show();
Help->hide();
}

Fl_Double_Window* make_help_window() {
  { Help = new Fl_Double_Window(300, 250);
    { Fl_Button* o = new Fl_Button(105, 210, 70, 20, "Got It!");
      o->callback((Fl_Callback*)cb_Got);
```

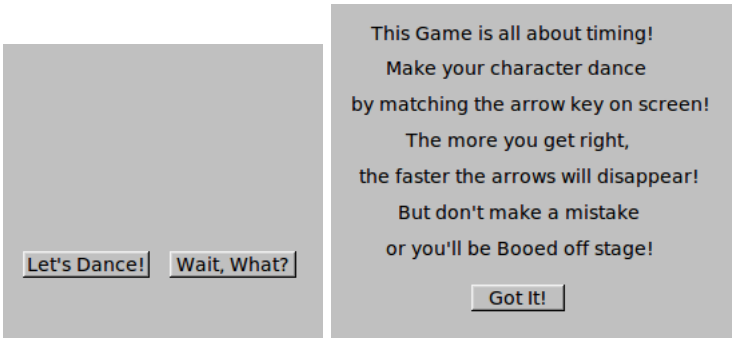
```
    } // Fl_Button* o
    { new Fl_Output(249, 10, 0, 25, "This Game is all about timing
    } // Fl_Output* o
    { new Fl_Output(239, 36, 0, 25, "Make your character dance");
    } // Fl_Output* o
    { new Fl_Output(287, 63, 0, 25, "by matching the arrow key on
    } // Fl_Output* o
    { new Fl_Output(226, 90, 0, 25, "The more you get right,");
    } // Fl_Output* o
    { new Fl_Output(279, 117, 0, 25, "the faster the arrows will d
    } // Fl_Output* o
    { new Fl_Output(234, 144, 0, 25, "But don\'t make a mistake");
    } // Fl_Output* o
    { new Fl_Output(245, 171, 0, 25, "or you\'ll be Booed off stag
    } // Fl_Output* o
    Help->end();
  } // Fl_Double_Window* Help
  return Help;
}

Game_Window* make_game_window() {
  return new Game_Window(500,500,"Dancing Game");
}
```

```
Fl_Double_Window *Main=(Fl_Double_Window *)0;

static void cb_Let(Fl_Button*, void*) {
  //Game_Window::position(200,200);
//Game_Window::show();
loadGameWindow();
Main->hide();
}

static void cb_Wait(Fl_Button*, void*) {
  Help->position(200,200);
Help->show();
Main->hide();
}

Fl_Double_Window* make_menu_window() {
  { Main = new Fl_Double_Window(240, 220, "Main Menu");
    { Fl_Button* o = new Fl_Button(15, 155, 95, 20, "Let\'s Dance!'
      o->callback((Fl_Callback*)cb_Let);
    } // Fl_Button* o
    { Fl_Button* o = new Fl_Button(125, 155, 95, 20, "Wait, What?"
      o->callback((Fl_Callback*)cb_Wait);
    } // Fl_Button* o
    Main->end();
```

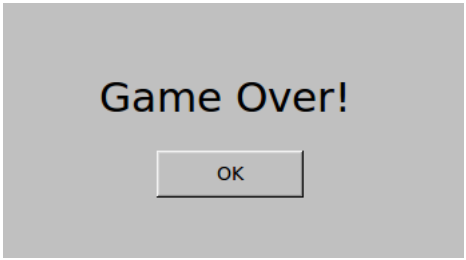
```
  } // Fl_Double_Window* Main
  return Main;
}

Fl_Double_Window *GameOver=(Fl_Double_Window *)0;

static void cb_OK(Fl_Button*, void*) {
  //gameOverOK();
GameOver->hide();

//Main->show();
}

Fl_Double_Window* make_gameover_window() {
  { GameOver = new Fl_Double_Window(345, 190);
    { Fl_Output* o = new Fl_Output(263, 58, 0, 25, "Game Over!");
      o->box(FL_FLAT_BOX);
      o->labelsize(31);
    } // Fl_Output* o
    { Fl_Button* o = new Fl_Button(115, 110, 110, 35, "OK");
      o->callback((Fl_Callback*)cb_OK);
    } // Fl_Button* o
    GameOver->end();
  } // Fl_Double_Window* GameOver
```

```
    return GameOver;
}
```

Actual Output




Quit
Press Spacebar to start


Game Over!
OK

Proof it works