1. *Team Information:*
Group 4

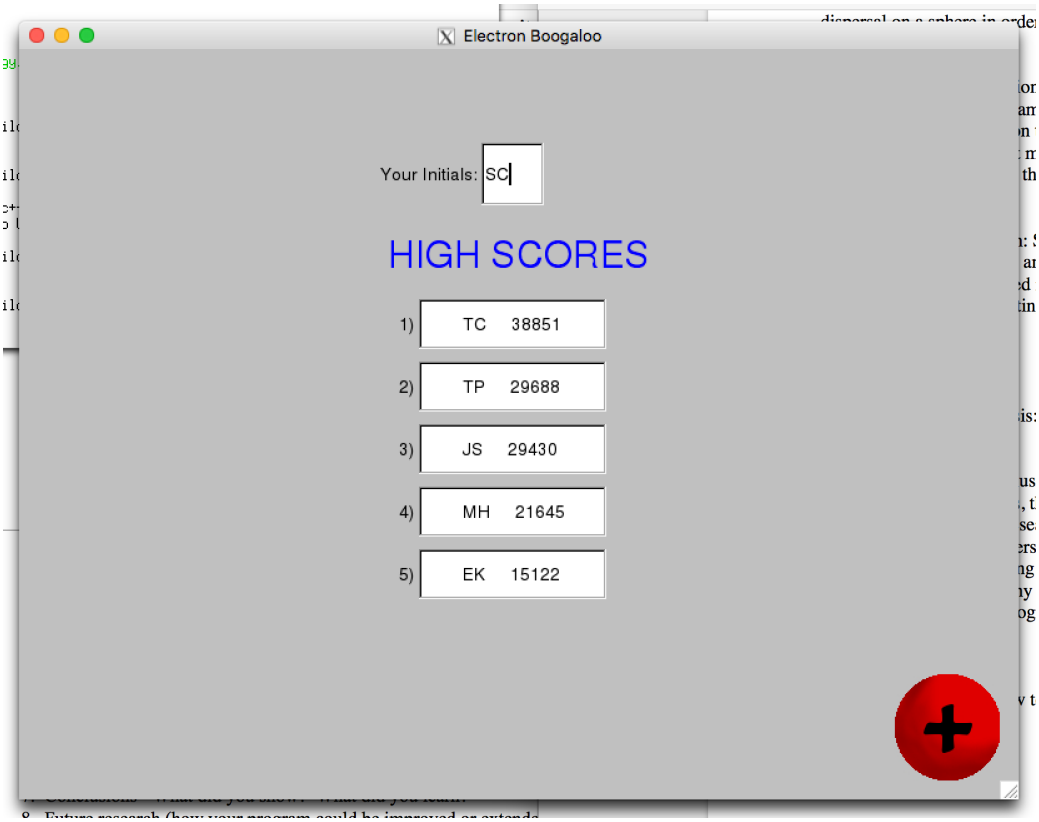
Tyler Carlson: Timer, Window creation, Initials input/output, Transitions
Joshua Crockett: Hint button, Distance calculations
Shehzad Charania Difficulty Settings, Timer
Travis DeAnda: Concept and Graphics (splash page, storyline/how-to, maps, etc.)

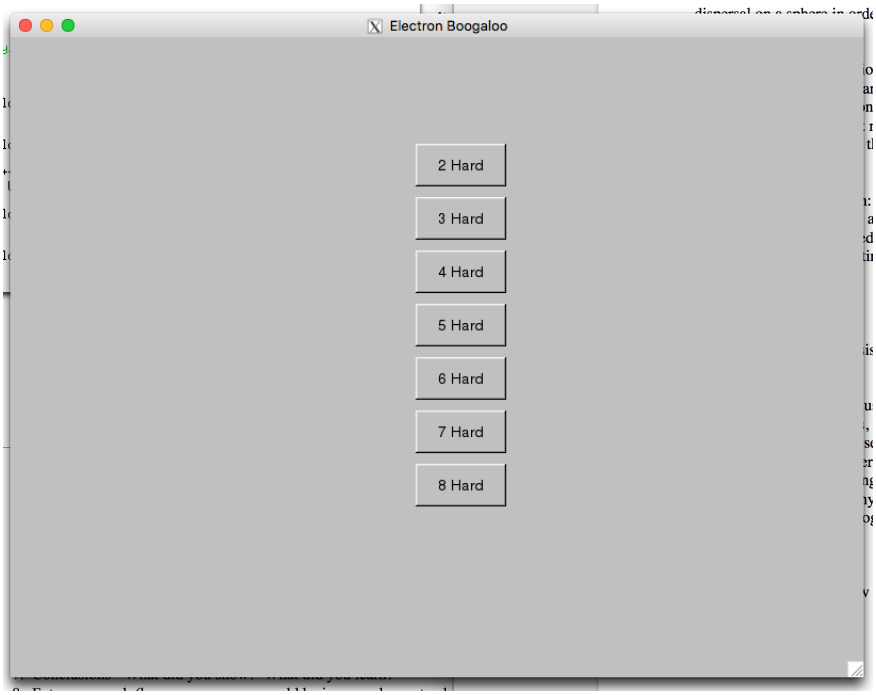
2. *Statement of the Problem:*
Design and write a C++14/FLTK computer game based on maximal dispersal on a sphere in order to determine where to place outer shell electrons most effectively and efficiently.

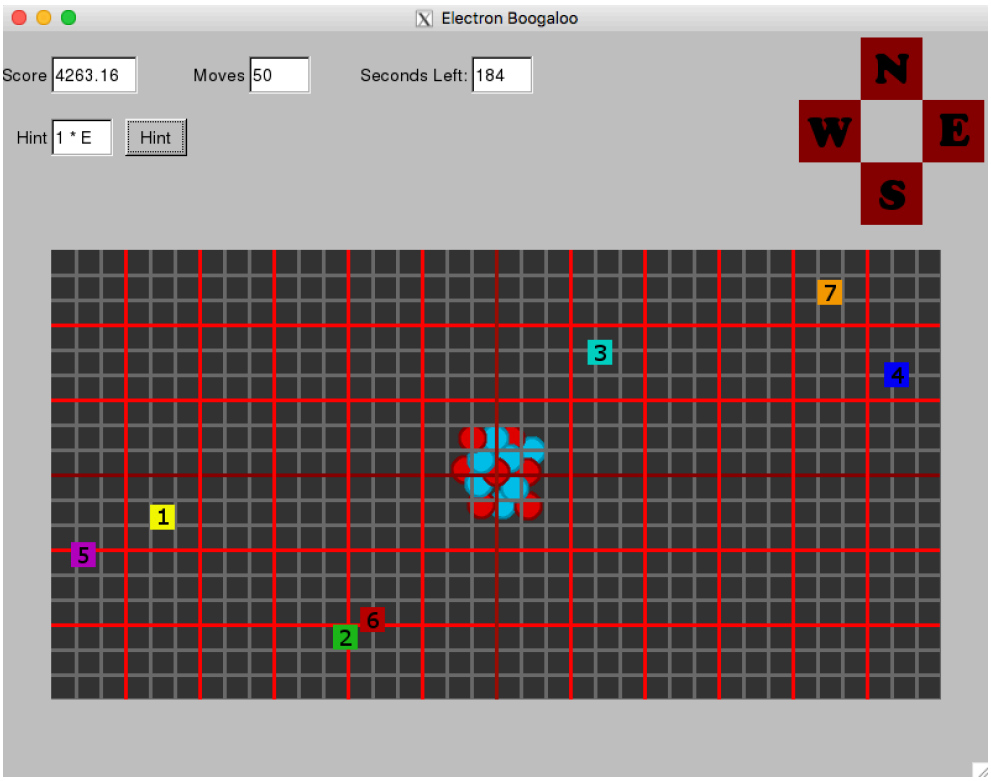
3. *Restrictions and Limitations:*
Generally creating a program such as this one should not be a problem for a veteran programmer, but unfortunately no one on this team is a seasoned programmer. To create this program, we had to actively search different methods and learn how to apply them to solve the program. These limitations did not impede the progress of our program nor did it impede the functionality of the program.

4. *Explanation of Approach:*
We started off sketching window layouts for various screens for game in order to determine buttons and boxes needed in program. The program was revised in PowerPoint with a sample color scheme added. We then determined the window size to begin creating graphics with proper dimensions to be added. Along with the window creation came attaching several buttons and output/input boxes.

5. *Sample Run:*







6. *Results and Analysis:*
The results of this project were very successful. As the implementation process continued along, each individual on our team learned more at a steady pace. The trickiest part of the program that was taken on was that of calculating the distance between each electron as well as creating a hint

button. Each electron had to be taken into account with its specific location, which ultimately made the concept a little difficult. However, throughout time all of the tasks became rather easier. Each task could be completed on within a timely manner, in which we still had 4 days left to spare after the project was finished. Within this time, we realized that the quit button would only work at random times and the timer would randomly stop at certain points. We fixed these issues and were all satisfied with the final product.

7. *Conclusions:*
The first and biggest conclusion that came to us is how difficult it is to maintain a program like this. For the majority of us, this was our first major programming project, so through this we learned how to properly research and implement programming ideas in our own code, whether that be through peer teachers or through various documentations online while still maintaining an organic code. The next thing that we learned was how to keep a code clean and well commented so that it's easy to mend any issues that might come along. Finally we learned how to tackle problems related to this program in other scenarios and future tasks if they come along.

8. *Future Research:*
We could have focused on creating a more succinct code that ran more efficiently, as well as making it more visually attractive. Because of the time period we had to complete our task, the visuals were not the most important thing, but can easily be improved on in the future. As for the code, the program as a whole could be written in a more sufficient manner, which will also shorten the length of the program.

9. *Instructions on how to run program:*
- Run the program
- Hit the next buttons
- Input initials
- Select difficulty
- Select electron and use the d-pad to move it around the screen to get the highest score possible
- The game ends when you run out of moves or when the timer runs out.
- Choose to play again or quit to program

10. *PROGRAM*

StartWinodw.cpp *(This file defines the window)*

```
#include "std_lib_facilities_4.h"
#include "Simple_window.h"
#include "Graph.h"
#include "Window.h"
#include "StartWindow.h"

//------------------------------------------------------------------------

// Constructors for Class/Struct

Start_window::Start_window(Point xy, int w, int h, const string& title):
    Graph_lib::Window(xy,w,h,title),
    next1_button(Point(x_max()-100,y_max()-100),85,85,"Next", [](Address,Address pw){reference_to<Start_window>(pw).next1();}),
    next2_button(Point(x_max()-100,y_max()-100),85,85,"Next", [](Address,Address pw){reference_to<Start_window>(pw).next2();}),
    D2_button(Point(x_max()/2-20,100),85,40,"2 Hard", [](Address,Address pw){reference_to<Start_window>(pw).D2();}),
    D3_button(Point(x_max()/2-20,150),85,40,"3 Hard", [](Address,Address pw){reference_to<Start_window>(pw).D3();}),
```

```
    D4_button(Point(x_max()/2-20,200),85,40,"4 Hard", [](Address,Address pw){reference_to<Start_window>(pw).D4();}),
    D5_button(Point(x_max()/2-20,250),85,40,"5 Hard", [](Address,Address pw){reference_to<Start_window>(pw).D5();}),
    D6_button(Point(x_max()/2-20,300),85,40,"6 Hard", [](Address,Address pw){reference_to<Start_window>(pw).D6();}),
    D7_button(Point(x_max()/2-20,350),85,40,"7 Hard", [](Address,Address pw){reference_to<Start_window>(pw).D7();}),
    D8_button(Point(x_max()/2-20,400),85,40,"8 Hard", [](Address,Address pw){reference_to<Start_window>(pw).D8();}),
    North_button(Point(x_max()-105,5),50,50,"", [](Address,Address pw){reference_to<Start_window>(pw).North();}),
    South_button(Point(x_max()-105,105),50,50,"", [](Address,Address pw){reference_to<Start_window>(pw).South();}),
    West_button(Point(x_max()-155,55),50,50,"", [](Address,Address pw){reference_to<Start_window>(pw).West();}),
    East_button(Point(x_max()-55,55),50,50,"", [](Address,Address pw){reference_to<Start_window>(pw).East();}),
    initial_ibox(Point(x_max()/2-30,75),50,50,"Your Initials:"),
    Score_1(Point(x_max()/2-80,200),150,40,"1)"),
    Score_2(Point(x_max()/2-80,250),150,40,"2)"),
    Score_3(Point(x_max()/2-80,300),150,40,"3)"),
    Score_4(Point(x_max()/2-80,350),150,40,"4)"),
    Score_5(Point(x_max()/2-80,400),150,40,"5)"),
    high_scores(Point(x_max()/2-105,175), "HIGH SCORES"),
    start_screen(Point(0,0),"Splash.jpg"),
    instructions(Point(0,0),"Storyline.jpg"),
    next_button(Point(x_max()-100,y_max()-100),"Button.jpg"),
    DPad(Point(x_max()-155,5),"DPad.jpg"),
    quit_button(Point(x_max()/2-75,y_max()/2+20),100,60,"QUIT", [](Address,Address pw){reference_to<Start_window>(pw).quit();}),
    play_again_button(Point(x_max()/2+75,y_max()/2+20),100,60,"PLAY AGAIN", [](Address,Address
pw){reference_to<Start_window>(pw).play_again();}),
    timer_box(Point(x_max()/2-20,20),50,30,"Seconds Left:"),
    HeliumGrids(Point(40,175),"HeliumGrids.jpg"),
    LithiumGrids(Point(40,175),"LithiumGrids.jpg"),
    BerylliumGrids(Point(40,175),"BerylliumGrids.jpg"),
    BoronGrids(Point(40,175),"BoronGrids.jpg"),
    CarbonGrids(Point(40,175),"CarbonGrids.jpg"),
    NitrogenGrids(Point(40,175),"NitrogenGrids.jpg"),
    OxygenGrids(Point(40,175),"OxygenGrids.jpg"),
    current_score(Point(40,20),70,30,"Score"),
    move_box(Point(200,20),50,30,"Moves"),
    hint_box(Point(40,70),50,30,"Hint"),
    hint_button(Point(100,70),50,30,"Hint", [](Address,Address pw){reference_to<Start_window>(pw).hint_help();}),
    button_pushed(false)
{
    attach(start_screen);
    attach(next_button);
    attach(next1_button);
    high_scores.set_font_size(30);
    high_scores.set_color(Color::blue);
}

Score::Score(string n, double s)
    :initials{n}, scores{s} {}


//------------------------------------------------------------------------------

// Operator Overloads

istream& operator>>(istream& input_score, Score& s){
    input_score >> s.initials >> s.scores;
    return input_score;
}

ostream & operator<<(ostream& output_score, Score& s){
    return output_score << s.initials << " " << s.scores << endl;
}

// Used to sort the scores
bool operator<(const Score& p1, const Score& p2){
    return (p1.scores < p2.scores);
}

//------------------------------------------------------------------------------

// Call Back Functions
```

```
void Start_window::cb_select_electron1(Address,Address pw) {
    reference_to<Start_window>(pw).select_electron1();
}

void Start_window::cb_select_electron2(Address,Address pw) {
    reference_to<Start_window>(pw).select_electron2();
}

void Start_window::cb_select_electron3(Address,Address pw) {
    reference_to<Start_window>(pw).select_electron3();
}

void Start_window::cb_select_electron4(Address,Address pw) {
    reference_to<Start_window>(pw).select_electron4();
}

void Start_window::cb_select_electron5(Address,Address pw) {
    reference_to<Start_window>(pw).select_electron5();
}

void Start_window::cb_select_electron6(Address,Address pw) {
    reference_to<Start_window>(pw).select_electron6();
}

void Start_window::cb_select_electron7(Address,Address pw) {
    reference_to<Start_window>(pw).select_electron7();
}

void Start_window::cb_select_electron8(Address,Address pw) {
    reference_to<Start_window>(pw).select_electron8();
}

//------------------------------------------------------------------------------

// Define wait for button

bool Start_window::wait_for_button(){
    show();
    button_pushed = false;
#if 1
    while (!button_pushed) Fl::wait();
    Fl::redraw();
#else
    Fl::run();
#endif
    return button_pushed;
}

//------------------------------------------------------------------------------

// Declarations/Definitions for variables in global scope

vector<Electron> calc_elec;

vector<Score> all_information; //Vector that contains scores of everyone

auto player_score {0};  //Final Score of Current Player and use of "auto"

int total_moves {0}; // Keeps track of total electron moves

int end_num {0}; // Goes to final screen when end_num = 1

int degree_moved {0}; // Keeps track of how many degrees to move

Score Current_player;

int tick_count {0}; // Keeps track of number of seconds

int num_ticks {0};  // Determines how many seconds to start the clock at
```

```
//------------------------------------------------------------------------

//Functions

// Create Timer
void Start_window::put_timer(){
    timer_box.put(to_string(num_ticks));
    redraw();
}

void Start_window::cb_timer(Address pw){
    ++tick_count;
    --num_ticks;
    reference_to<Start_window>(pw).put_timer();
    if(tick_count < calc_elec.size()*30+1){
        Fl::repeat_timeout(1.0,cb_timer, pw);
    }
    if(num_ticks == 0) {
        reference_to<Start_window>(pw).actions_end();
    }
}

void Start_window::timer(){
    Fl::add_timeout(1.0, cb_timer, this);
}

// Create a string of scores with initials
string create_string(vector<Score>& all_scores, int slot){
    ostringstream os;
    os <<setw(10) <<all_scores[slot].initials << setw(10) << all_scores[slot].scores;
    return os.str();
}

// Put all of the scores from the file into a vector
void read_score_to_game(vector<Score>& all_scores, const string& iname){
    ifstream ist {iname};
    if(!ist) error("Can't open input file ", iname);
    for(Score player; ist >> player;){
        all_scores.push_back(player);
    }
    ist.close();
}

// Write Player's Score to the file for storage
void write_score_to_file(Score p1,double final_score ,vector<Score>& all_scores, const string& oname){
    p1.scores = final_score;
    ofstream ost {oname};
    if(!ost) error("Can't open output file ", oname);
    all_scores.push_back(p1);
    for(auto i: all_scores){    // Range Based For Loop and "auto"
        ost << i << "\n";
    }
    ost.close();
}

//distance calculations between 2 electrons
double calc_dist(Electron s1, Electron s2, double radius) {
    return radius*acos( sin(s1.latitude*M_PI/180)*sin(s2.latitude*M_PI/180)+
    cos(s1.latitude*M_PI/180)*cos(s2.latitude*M_PI/180)*cos(abs(s1.longitude-s2.longitude)*M_PI/180));
}

//distance calculations between each electrons put in vector and sorted, and smallest element outputted
double shortest_distance(vector<Electron> v) {
    vector<double> values;
    for (int i=0; i<v.size(); ++i) {
        for (int j=i+1; j<v.size(); ++j) {
            values.push_back(calc_dist(v[i],v[j],4000));
        }
    }
    sort(values.begin(), values.end());
```

```
    return values[0];
}

//initializes start spots for electron
void random_start(vector<Electron>& v) {
    srand(time(NULL));
    for (Electron& i: v) {  // use of range for loop
        int rnum=rand();
        int plong=rand()%181;
        int plat=rand()%91;
        i.longitude=pow(-1,rnum)*plong;//180+-
        i.latitude=pow(-1,rnum)*plat;//90+-
    }
}

//keeps electonss on map and deals with wrap-around
void bound_check(Electron& e) {
    if(e.latitude>90) {
        e.latitude=180-e.latitude;
        e.longitude+=180;
    }
    else if(e.latitude<-90) {
        e.latitude=-180-e.latitude;
        e.longitude+=180;
    }
    if(e.longitude>180) {
        e.longitude-=360;
    }
    else if(e.longitude<-180) {
        e.longitude+=360;
    }
}

//tells electron the distance to move on map in terms of pixels
int x_dist_move (int n,  Electron e) {
    return 2*(e.longitude-n);
}

int y_dist_move (int n, Electron e) {
    return 2*(n-e.latitude);
}

//places electron in correct initial spot
int x_convert (Electron e) {
    return 390+2*e.longitude;
}

int y_convert (Electron e) {
    return 345-2*e.latitude;
}

//each hint_vect_ adds the shortest distance after a move in named direction (N,E,S,W) to a vector
void hint_vect_e(vector<Electron> e, vector<double>& vv,int n){
    vector<Electron> bb8;
    bb8=e;
    for (int i=0; i<e.size();++i) {
        bb8[i].longitude+=n;
        bound_check(bb8[i]);
        vv.push_back(shortest_distance(bb8));
        bb8[i].longitude=e[i].longitude;
    }
}

void hint_vect_w(vector<Electron> e, vector<double>& vv,int n) {
    vector<Electron> bb8;
    bb8=e;
    for (int i=0; i<e.size();++i) {
        bb8[i].longitude-=n;
        bound_check(bb8[i]);
        vv.push_back(shortest_distance(bb8));
```

```
        bb8[i].longitude=e[i].longitude;
    }
}

void hint_vect_n(vector<Electron> e, vector<double>& vv,int n) {
    vector<Electron> bb8;
    bb8=e;
    for (int i=0; i<e.size();++i) {
        bb8[i].latitude+=n;
        bound_check(bb8[i]);
        vv.push_back(shortest_distance(bb8));
        bb8[i].latitude=e[i].latitude;
    }
}

void hint_vect_s(vector<Electron> e, vector<double>& vv,int n) {
    vector<Electron> bb8;
    bb8=e;
    for (int i=0; i<e.size();++i) {
        bb8[i].latitude-=n;
        bound_check(bb8[i]);
        vv.push_back(shortest_distance(bb8));
        bb8[i].latitude=e[i].latitude;
    }
}

//function collects all the vectors from the directions into one vector
void hint_vect(vector<Electron> eei, vector<double>& vvi,int nni) {
    hint_vect_e(eei,vvi,nni);
    hint_vect_w(eei,vvi,nni);
    hint_vect_n(eei,vvi,nni);
    hint_vect_s(eei,vvi,nni);
}

//function detrmines which move was best by its position in vector and displays it to outbox
void Start_window::hint_help() {
    vector<double> v (0);
    hint_vect(calc_elec,v,degree_moved);
    vector<double> vect;
    vect=v;
    char ch;
    bool valfond=true;
    sort(vect.begin(), vect.end());
    for (int i=0; i<v.size() && valfond; ++i) {
        if (v[i]==vect[vect.size()-1]) {
            if(i/calc_elec.size()<1) {ch='E';}
            else if(i/calc_elec.size()>=1 && i/calc_elec.size()<2) {ch='W';}
            else if(i/calc_elec.size()>=2 && i/calc_elec.size()<3) {ch='N';}
            else if(i/calc_elec.size()>=3) {ch='S';}
            ostringstream sout;
            sout<<""<<i%calc_elec.size()+1<<" * "<<ch;
            string ss=sout.str();
            hint_box.put(ss);
            valfond=false;
        }
    }
}

// Actions to be completed on second next button click
void Start_window::actions_2(){
    detach(next1_button);
    detach(instructions);
    attach(initial_ibox);
    attach(next2_button);
    attach(high_scores);
    attach(Score_1);
    attach(Score_2);
    attach(Score_3);
    attach(Score_4);
    attach(Score_5);
```

```
    redraw();
}

// Actions to be completed on third next button click
void Start_window::actions_3(){
    detach(initial_ibox);
    detach(next2_button);
    detach(high_scores);
    detach(Score_1);
    detach(Score_2);
    detach(Score_3);
    detach(Score_4);
    detach(Score_5);
    attach(D2_button);
    attach(D3_button);
    attach(D4_button);
    attach(D5_button);
    attach(D6_button);
    attach(D7_button);
    attach(D8_button);
    detach(next_button);
}

//for reading out score and moves left
void Start_window::score_move_output() {
    ostringstream sout,s2out;
    sout<<calc_elec.size()*shortest_distance(calc_elec);
    s2out<<50-total_moves;
    string ss=sout.str();
    string ss2=s2out.str();
    current_score.put(ss);
    move_box.put(ss2);
}

// Actions to be completed when difficulty button is clicked
void Start_window::actions_general_menu(){
    detach(D2_button);
    detach(D3_button);
    detach(D4_button);
    detach(D5_button);
    detach(D6_button);
    detach(D7_button);
    detach(D8_button);
    attach(North_button);
    attach(South_button);
    attach(West_button);
    attach(East_button);
    attach(DPad);
    attach(current_score);
    attach(move_box);
    attach(hint_box);
    attach(hint_button);
    for(int i = 0;i < electron_v.size();++i) attach(electron_v[i]);
    for(int i = 0;i < electron_image.size();++i) attach(electron_image[i]);
    attach(timer_box);
    timer();
    score_move_output();
}

// Detach these items in the actions end function
void Start_window::detach_items(){
    detach(North_button);
    detach(South_button);
    detach(West_button);
    detach(East_button);
    detach(DPad);
    detach(HeliumGrids);
    detach(LithiumGrids);
    detach(BerylliumGrids);
    detach(BoronGrids);
```

```
    detach(CarbonGrids);
    detach(NitrogenGrids);
    detach(OxygenGrids);
    detach(timer_box);
    detach(move_box);
    detach(hint_box);
    detach(hint_button);
}

// Actions to be completed when game is over (time is out or out of moves)
void Start_window::actions_end(){
    detach_items();
    current_score.move(360,200);
    for(int i = 0;i < electron_v.size();++i) detach(electron_v[i]);
    for(int i = 0;i < electron_image.size();++i) detach(electron_image[i]);
    attach(quit_button);
    attach(play_again_button);
    player_score = calc_elec.size()*shortest_distance(calc_elec);
    write_score_to_file(Current_player,player_score,all_information,"all_scores.txt");
    Fl::remove_timeout(cb_timer);
}

// Put top 5 scores from vector into outboxes
void Start_window::read_out_scores() {
    if (all_information.size()==0){}//if file is less than 5, reads out as
    if (all_information.size()>0){// many as possible
        Score_1.put(create_string(all_information,all_information.size()-1));
    }
    if (all_information.size()>1){
        Score_2.put(create_string(all_information,all_information.size()-2));
    }
    if (all_information.size()>2){
        Score_3.put(create_string(all_information,all_information.size()-3));
    }
    if (all_information.size()>3){
        Score_4.put(create_string(all_information,all_information.size()-4));
    }
    if (all_information.size()>4){
        Score_5.put(create_string(all_information,all_information.size()-5));
    }
}

//creates buttons that represent electrons
void Start_window::create_electrons(int i) {
    if (i>1) {
        electron_v.push_back(new Button(Point{x_convert(calc_elec[0]),y_convert(calc_elec[0])},20,20,"",cb_select_electron1));
        electron_v.push_back(new Button(Point{x_convert(calc_elec[1]),y_convert(calc_elec[1])},20,20,"",cb_select_electron2));
        }
    if (i>2) {electron_v.push_back(new Button(Point{x_convert(calc_elec[2]),y_convert(calc_elec[2])},20,20,"",cb_select_electron3));}
    if (i>3) {electron_v.push_back(new Button(Point{x_convert(calc_elec[3]),y_convert(calc_elec[3])},20,20,"",cb_select_electron4));}
    if (i>4) {electron_v.push_back(new Button(Point{x_convert(calc_elec[4]),y_convert(calc_elec[4])},20,20,"",cb_select_electron5));}
    if (i>5) {electron_v.push_back(new Button(Point{x_convert(calc_elec[5]),y_convert(calc_elec[5])},20,20,"",cb_select_electron6));}
    if (i>6) {electron_v.push_back(new Button(Point{x_convert(calc_elec[6]),y_convert(calc_elec[6])},20,20,"",cb_select_electron7));}
    if (i>7) {electron_v.push_back(new Button(Point{x_convert(calc_elec[7]),y_convert(calc_elec[7])},20,20,"",cb_select_electron8));}
}

//creates images for the buttons that represent electrons
void Start_window::create_electrons_jpg(int i) {
    if (i>1) {
        electron_image.push_back(new Image(Point{x_convert(calc_elec[0]),y_convert(calc_elec[0])},"Electron1.jpg"));
        electron_image.push_back(new Image(Point{x_convert(calc_elec[1]),y_convert(calc_elec[1])},"Electron2.jpg"));
    }
    if (i>2) {electron_image.push_back(new Image(Point{x_convert(calc_elec[2]),y_convert(calc_elec[2])},"Electron3.jpg"));}
    if (i>3) {electron_image.push_back(new Image(Point{x_convert(calc_elec[3]),y_convert(calc_elec[3])},"Electron4.jpg"));}
    if (i>4) {electron_image.push_back(new Image(Point{x_convert(calc_elec[4]),y_convert(calc_elec[4])},"Electron5.jpg"));}
    if (i>5) {electron_image.push_back(new Image(Point{x_convert(calc_elec[5]),y_convert(calc_elec[5])},"Electron6.jpg"));}
    if (i>6) {electron_image.push_back(new Image(Point{x_convert(calc_elec[6]),y_convert(calc_elec[6])},"Electron7.jpg"));}
    if (i>7) {electron_image.push_back(new Image(Point{x_convert(calc_elec[7]),y_convert(calc_elec[7])},"Electron8.jpg"));}
}
```

```
//makes the 10 steps 20,10,5,1 degrees
void calculate_moves(){
    if (total_moves < 10) degree_moved = 20;
    if (total_moves >= 10 && total_moves < 20) degree_moved = 15;
    if (total_moves >= 20 && total_moves < 30) degree_moved = 10;
    if (total_moves >= 30 && total_moves < 40) degree_moved = 5;
    if (total_moves >= 40 && total_moves < 50) degree_moved = 1;
    if (total_moves == 49) end_num = 1;
}

// Reset appropriate variables for play again
void reset_variables(){
    tick_count =0;
    num_ticks = 0;
    total_moves = 0;
    end_num = 0;
    degree_moved = 0;
    all_information.clear();
}

//-------------------------------------------------------------------------

//Functions used by Call back functions

// First and Second button call back functions
void Start_window::next1(){
    ++button_count;
    if(button_count == 1){
        detach(start_screen);
        attach(instructions);
        attach(next_button);
        redraw();
    } else if(button_count == 2){
        actions_2();
        read_score_to_game(all_information,"all_scores.txt");
        sort(all_information.begin(), all_information.end());
        read_out_scores();
    }
}

// Third button call back function
void Start_window::next2(){
    string player_initial = initial_ibox.get_string();
    Current_player.initials = player_initial; // Initialize Current Player Initials
    actions_3();
}

//select_electron 1-8 are the button functions for the electrons
//tells which electron has been selected
void Start_window::select_electron1(){
    move_electron=0;
}

void Start_window::select_electron2(){
    move_electron=1;
}

void Start_window::select_electron3(){
    move_electron=2;
}

void Start_window::select_electron4(){
    move_electron=3;
}

void Start_window::select_electron5(){
    move_electron=4;
}

void Start_window::select_electron6(){
```

```
    move_electron=5;
}

void Start_window::select_electron7(){
    move_electron=6;
}

void Start_window::select_electron8(){
    move_electron=7;
}

//D2-8 are difficulty level functions, and puts the right amount of time on the timer and
//electrons on the screen
void Start_window::D2(){
    num_ticks = 2*30+1;
    calc_elec.resize(2);
    random_start(calc_elec);
    create_electrons(2);
    create_electrons_jpg(2);
    attach(HeliumGrids);
    actions_general_menu();
}

void Start_window::D3(){
    num_ticks = 3*30+1;
    calc_elec.resize(3);
    random_start(calc_elec);
    create_electrons(3);
    create_electrons_jpg(3);
    attach(LithiumGrids);
    actions_general_menu();
}

void Start_window::D4(){
    num_ticks = 4*30+1;
    calc_elec.resize(4);
    random_start(calc_elec);
    create_electrons(4);
    create_electrons_jpg(4);
    attach(BerylliumGrids);
    actions_general_menu();
}

void Start_window::D5(){
    num_ticks = 5*30+1;
    calc_elec.resize(5);
    random_start(calc_elec);
    create_electrons(5);
    create_electrons_jpg(5);
    attach(BoronGrids);
    actions_general_menu();
}

void Start_window::D6(){
    num_ticks = 6*30+1;
    calc_elec.resize(6);
    random_start(calc_elec);
    create_electrons(6);
    create_electrons_jpg(6);
    attach(CarbonGrids);
    actions_general_menu();
}

void Start_window::D7(){
    num_ticks = 7*30+1;
    calc_elec.resize(7);
    random_start(calc_elec);
    create_electrons(7);
    create_electrons_jpg(7);
    attach(NitrogenGrids);
```

```
    actions_general_menu();
}

void Start_window::D8(){
    num_ticks = 8*30+1;
    calc_elec.resize(8);
    random_start(calc_elec);
    create_electrons(8);
    create_electrons_jpg(8);
    attach(OxygenGrids);
    actions_general_menu();
}

//movement operations for the electrons
void Start_window::North(){
    calculate_moves();
    ++total_moves;
    int a=calc_elec[move_electron].longitude;
    int b=calc_elec[move_electron].latitude;
    calc_elec[move_electron].latitude+=degree_moved;
    bound_check(calc_elec[move_electron]);
    electron_v[move_electron].move(x_dist_move(a,calc_elec[move_electron]),y_dist_move(b,calc_elec[move_electron]));
    electron_image[move_electron].move(x_dist_move(a,calc_elec[move_electron]),y_dist_move(b,calc_elec[move_electron]));
    score_move_output();
    hint_box.put("");
    if(end_num == 1) actions_end();
}

void Start_window::South(){
    calculate_moves();
    ++total_moves;
    int a=calc_elec[move_electron].longitude;
    int b=calc_elec[move_electron].latitude;
    calc_elec[move_electron].latitude-=degree_moved;
    bound_check(calc_elec[move_electron]);
    electron_v[move_electron].move(x_dist_move(a,calc_elec[move_electron]),y_dist_move(b,calc_elec[move_electron]));
    electron_image[move_electron].move(x_dist_move(a,calc_elec[move_electron]),y_dist_move(b,calc_elec[move_electron]));
    score_move_output();
    hint_box.put("");
    if(end_num == 1) actions_end();
}

void Start_window::West(){
    calculate_moves();
    ++total_moves;
    int a=calc_elec[move_electron].longitude;
    int b=calc_elec[move_electron].latitude;
    calc_elec[move_electron].longitude-=degree_moved;
    bound_check(calc_elec[move_electron]);
    electron_v[move_electron].move(x_dist_move(a,calc_elec[move_electron]),y_dist_move(b,calc_elec[move_electron]));
    electron_image[move_electron].move(x_dist_move(a,calc_elec[move_electron]),y_dist_move(b,calc_elec[move_electron]));
    score_move_output();
    hint_box.put("");
    if(end_num == 1) actions_end();
}

void Start_window::East(){
    calculate_moves();
    ++total_moves;
    int a=calc_elec[move_electron].longitude;
    int b=calc_elec[move_electron].latitude;
    calc_elec[move_electron].longitude+=degree_moved;
    bound_check(calc_elec[move_electron]);
    electron_v[move_electron].move(x_dist_move(a,calc_elec[move_electron]),y_dist_move(b,calc_elec[move_electron]));
    electron_image[move_electron].move(x_dist_move(a,calc_elec[move_electron]),y_dist_move(b,calc_elec[move_electron]));
    score_move_output();
    hint_box.put("");
    if(end_num == 1) actions_end();
}
```

```
//end game options
void Start_window::quit(){
    button_pushed = true;
    exit(0);
}

void Start_window::play_again(){
    hide();
    reset_variables();
    Start_window win{Point(100,100),800,600,"Electron Boogaloo"};
    win.wait_for_button();
}
```

StartWindow.h *(This file creates all classes/structs with functions/items to be used)*

```
#include "std_lib_facilities_4.h"
#include "Simple_window.h"
#include "Graph.h"
#include "Window.h"


struct Start_window : Graph_lib::Window {
    Start_window(Point xy, int w, int h, const string& title);
    int button_count {0}; // Keeps track of how many times the first button is clicked
    int move_electron {0}; // Keeps track of which electron is clicked to move

private:
    // Declaration of Buttons
    Button next1_button;
    Button next2_button;
    Button D2_button;
    Button D3_button;
    Button D4_button;
    Button D5_button;
    Button D6_button;
    Button D7_button;
    Button D8_button;
    Button North_button;
    Button South_button;
    Button West_button;
    Button East_button;
    Button quit_button;
    Button play_again_button;
    Button hint_button;

    Vector_ref<Button> electron_v;  // Vector that contains Buttons for electrons
    Vector_ref<Image> electron_image; // Vector that contains Images for electrons

    Text high_scores; // "High Scores" text

    // Declaration of Images
    Image start_screen;
    Image instructions;
    Image next_button;
    Image DPad;
    Image HeliumGrids;
    Image LithiumGrids;
    Image BerylliumGrids;
    Image BoronGrids;
    Image CarbonGrids;
    Image NitrogenGrids;
    Image OxygenGrids;

    In_box initial_ibox; // Box declared to input initials

    // Declaration of Out Boxes
    Out_box Score_1;
    Out_box Score_2;
    Out_box Score_3;
    Out_box Score_4;
```

```
    Out_box Score_5;
    Out_box timer_box;
    Out_box current_score;
    Out_box move_box;
    Out_box hint_box;

    // Used for play again
    bool wait_for_button();
    bool button_pushed;

    // Functions used by called back functions
    void next1();
    void next2();
    void D2();
    void D3();
    void D4();
    void D5();
    void D6();
    void D7();
    void D8();
    void select_electron1();
    void select_electron2();
    void select_electron3();
    void select_electron4();
    void select_electron5();
    void select_electron6();
    void select_electron7();
    void select_electron8();
    void quit();
    void play_again();
    void put_timer();
    void North();
    void South();
    void West();
    void East();

    // Call back functions
    static void cb_select_electron1(Address, Address);
    static void cb_select_electron2(Address, Address);
    static void cb_select_electron3(Address, Address);
    static void cb_select_electron4(Address, Address);
    static void cb_select_electron5(Address, Address);
    static void cb_select_electron6(Address, Address);
    static void cb_select_electron7(Address, Address);
    static void cb_select_electron8(Address, Address);
    static void cb_timer(Address);

    // Declaration of functions used throughout program
    void actions_2();
    void actions_3();
    void actions_general_menu();
    void actions_end();
    void read_out_scores();
    void create_electrons(int i);
    void create_electrons_jpg(int i);
    void timer();
    void score_move_output();
    void hint_help();
    void detach_items();

};

// Class containing information for each player
class Score{
public:
    string initials;
    double scores;

    Score() {};
    Score(string n, double s);
```

```
};

// Struct containing information for each electron
struct Electron {
   double longitude;
   double latitude;
};
```

main.cpp *(This file creates the StartWindow and runs the program)*

```
#include "std_lib_facilities_4.h"
#include "StartWindow.h"

int main(){
   Start_window win{Point(100,100),800,600,"Electron Boogaloo"};
   return gui_main();
}
```

11. *Bibliography:*

http://www.fltk.org/documentation.php