Project 3 Report

Joseph Gerules : Graphics (33.33%)

Yuxuan Liao : Input Output from File (33.33%)

Christopher Padilla : AI (33.33%)

Team 1

The Red One

Problem and Significance

The purpose of this project was to implement a simple version of a 3-dimensional 4x4x4 game of tic-tac-toe and thoroughly test its functionality as an application. Our team of developers has been working endlessly to ensure the user gains the best experience possible when playing this game.

As a group, we were able to develop, test and deliver a working product while managing a tight deadline and quickly gaining the skills necessary to be successful. The goal was to manage many different moving parts of project development (i.e. one person doing the AI and minimax algorithm and the other two managing what happens on the user's side).

Not only was this good practice in program development and understanding the concept rational agents in mathematical game theory, it is now also a fun game that anyone can enjoy, and whether they can program or not, the intent is to provide players with a sense of pride and accomplishment for executing and playing a fun game of tic-tac-toe.

Restrictions and Limitations

Due to company constraints, we were limited to programming the game in C++ and using a text-based set of libraries called ncurses for our user interface. Of course, these kinds of limits exist out here in the real world, and we are consciously prepared to deal with them as we have proven to ourselves that we are capable of doing so.

As mentioned above, we were required to meet the programming environment space limitations (C++ and ncurses). In addition, to these space limitations, we had *actual* space limitations we had to satisfy, primarily with the coding of the artificial agent that the user plays against. The full minimax tree space requirements for this problem is on the order of 10^{89} nodes. 10^{89} is a biblically large number. 10^{89} is more than the age of the universe in femtoseconds. 10^{89} is *very* close to the size of Jerry Jones' ego, and there is no way we'd able to create and traverse the full game tree without major advancements in quantum computing.

Not only were there programming space limitations as mentioned above, where the team was forced to create and traverse smaller minimax trees, there were

also programming *time* limitations. These understandably come across the whole discipline of computer science and game development, as the user likely does not want to wait forever on a machine, be it loading screens or, in our case, deciding the next move. Therefore, we had to carefully determine the optimal way our agent could play while adhering to the constraint of having the computer move within ten seconds of the user's input.

Overall Development Approach

During the first lab meeting, the team considered and analyzed its options and decided to assign smaller parts to individual members. The risky but ultimate consensus was to let each person do his own thing for the time allotted and meet up a couple of days before each deadline. The team begun by dividing the project into pieces that each member could complete separately. For the first deadline, Joseph created the board and game flow while Yuxuan managed the splash screen and scorekeeping. Christopher during this time started working on and tested the artificial agent and minimax tree; this part was started early because we assumed it would be the toughest to implement.

The final few days of development were spent on integrating all of the parts with one another. The AI coder carefully kept his code agile, object-oriented and mostly open-ended for assimilation with the user interface. Functions and methods such as convert_to_vector(), which converts a numeric cell to vector notation, allowed for smoother integration with the UI programmer's code. The scorekeeping and splash screen programmer also tightly conferred with the UI coder to ensure proper linkage as we developed towards a final product.

Production Notes

Design Choices for Framework Code 3-27-18

1. We used ncurses non-window functions since we had more code written in that format.
    - Using the same format of commands across the board was essential to the program actually working and caused a couple problems upon integrating the non-window code with the window code.
2. Decided to not put everything into classes, but simply to have a giant file.
    - This was easy to write and use for the first turn in, but meant that we had a kinda sloppy setup for our second turn in/integration.
    - All of our functions ended up being quite compartmentalized due to this and the lack of global variables which was actually a plus.
3. Make a red border for team morale.
    - It worked. 10/10 would make the border red again.

Design Choices for Final Code 4-3-18

1. Made only a couple places where the AI had to speak with the graphics.
    a. This made the integration “easy” after we had decided exactly what would be passed between the two sets of code. This meant we had more compartmentalized code.
2. Choosing the user interface coder’s programming flow
    a. Explanation: Both the UI and AI coders made their own basic program flow for testing and development, and we stuck the UI’s flow in the end. This made for less rewriting of code and simpler implementation.
3. AI code remained very agile and object-oriented.
    a. This got all of the AI methods out of the way of the main functions needed for the user interface.
    b. Everything is an object and made for very easy manipulation of data

---

General Knowledge Gained

1. More of a reminder, but the use of a refresh() like function vs simply having things show up on the screen created some learning moments.

2. You don’t actually have to be a super-genius to develop an algorithm that consistently wins games zero-sum games such as tic-tac-toe (even if you can’t win them yourself). The process is there and the math is always correct.

Future Research/Usage

1. Further explore the use of ncurses/better user interfaces
2. Multi-users game play
3. Cross platform capability
4. AI core will be used in project 4

Conclusion

Over the course of the past two weeks we were able to work together to reach the common goal of having a presentable implementation of the game. This was a good way to learn to *learn.* The use of ncurses and AI meant we had to learn quickly and adapt, which therefore makes us better programmers. Even if not all of us are going into game development, this was still still a very good exercise of the way we may have to end up working in the real world. It’s great to have all kinds of individual knowledge, but if you are unable to make use of it by committing to something as a group, then it is all for naught. We learned that we can be very successful by bringing our ideas and work initiatives together.

Instructions

To run the code:

1. Ensure Ncurses is installed (leaving that one to you).
2. Put all the files in the same folder and navigate to that folder.
3. Compile the program with:
    a. g++ -std=c++11 *.cpp -lncurses -Ofast
4. Now run your executable either by:
    a. ./a.out on linux or
    b. a.exe on windows
5. To play our game follow the prompts to enter a name, and choose your marker.
6. Playing the game is done by pressing the space key to place a mark.

Bibliography

"Stack Overflow - Where Developers Learn, Share, & Build Careers." *Stack Overflow - Where Developers Learn, Share, & Build Careers*, stackoverflow.com/.

NCURSES Programming HOWTO, tldp.org/HOWTO/NCURSES-Programming-HOWTO/.

Development Log

FIRST DUE DATE: 3/21/18

SECOND DUE DATE: 3/28/18

THIRD DUE DATE: 4/2/18

3/1/18 12:30 PM aggresive-stag, lyx0203, cpadilla

TEAMS ASSIGNED!

3/20/18 12:30 PM aggresive-stag

Setup github wikis and homepages(HOUSEKEEPING)

3/20/18 1:30 PM aggresive-stag, cpadilla

Started work on design document

aggresive-stag: high level entities, interaction, conclusion

cpadilla: opening remarks, low level entities

3/21/18 8:20 PM aggresive-stag, lyx0203, cpadilla

Finishing design document

lyx0203: problems we could run into

cpadilla: ER diagram using Microsoft Visio

aggresive-stag: Interaction flow diagram

3/27/18 10:00 AM aggresive-stag

Started work on the game

3/27/18 12:40 PM aggresive-stag, lyx0203, cpadilla

Met for lab and assigned what each person has to do

aggresive-stag: UI and gameflow

lyx0203: splash screen and scorekeeping

cpadilla: AI and minimax tree

3/27/18 2:20 PM aggresive-stag

Continued work on game.

3/27/18 6:20 PM aggresive-stag, cpadilla

Discussed the code.

aggresive-stag: finished cleaning up UI and splash screen

cpadilla: moral support

3/28/18 4:30 PM lyx0203

Added several functions to complete the splash screen.

3/28/18 6:20 PM aggresive-stag

Added some finishing touches to the bucket.

4/2/18 6:00 PM aggresive-stag, lyx0203, cpadilla

Met to integrate AI with graphics.

aggresive-stag: gave ideas as to how each other’s functions will be called

cpadilla: programmed the base function of the solution

lyx0203: continued cleaning up scorekeeping

4/2/18 9:00 PM aggresive-stag, cpadilla

Minor issues still giving issues :/

aggresive-stag: fixed several bugs due to UI integration with AI

cpadilla: continued bug fixing of middleman functions/methods

4/3/18 12:00 PM aggresive-stag, lyx0203, cpadilla

Met in lab. Spoke about file handling and minor bugs.

cpadilla: tested AI with UI

aggresive-stag: added features to game experience

lyx0203: continued work on scorekeeping

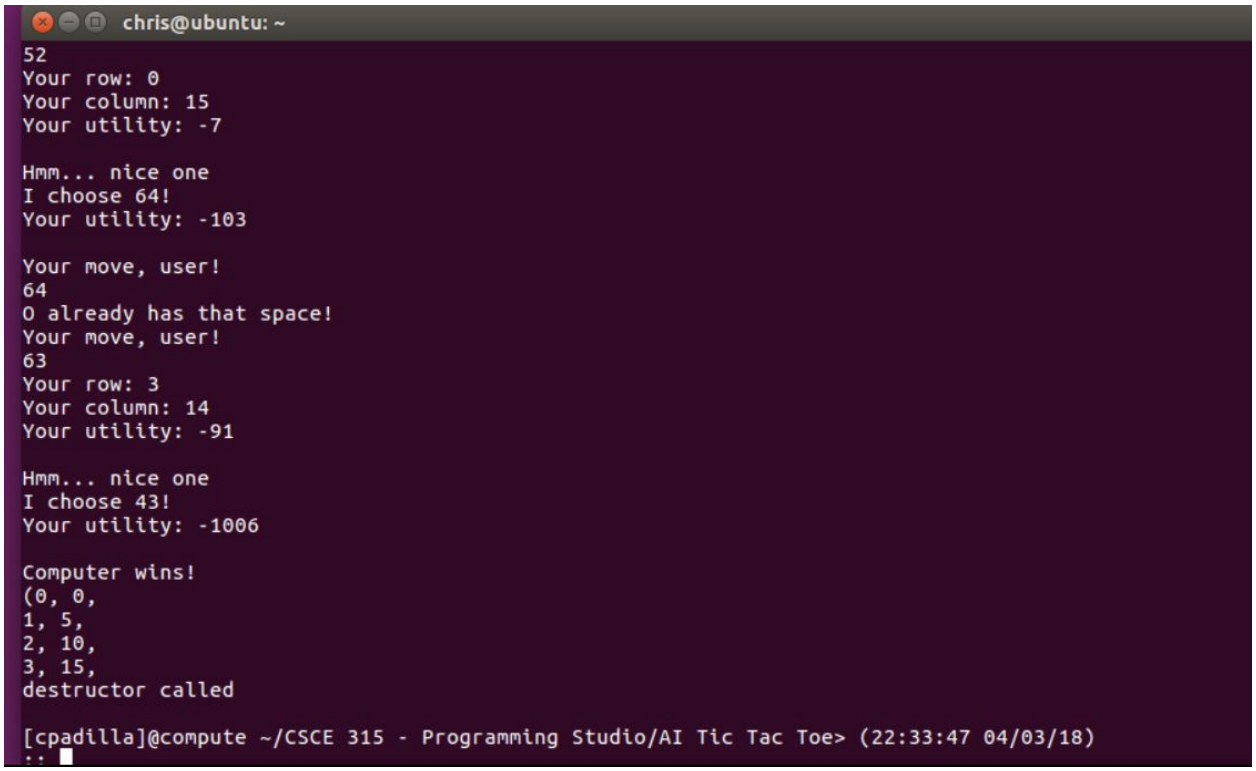
4/3/18 6:00 PM aggresive-stag, lyx0203, cpadilla

FINAL HAUL
aggresive-stag: added extra features to improve user experience

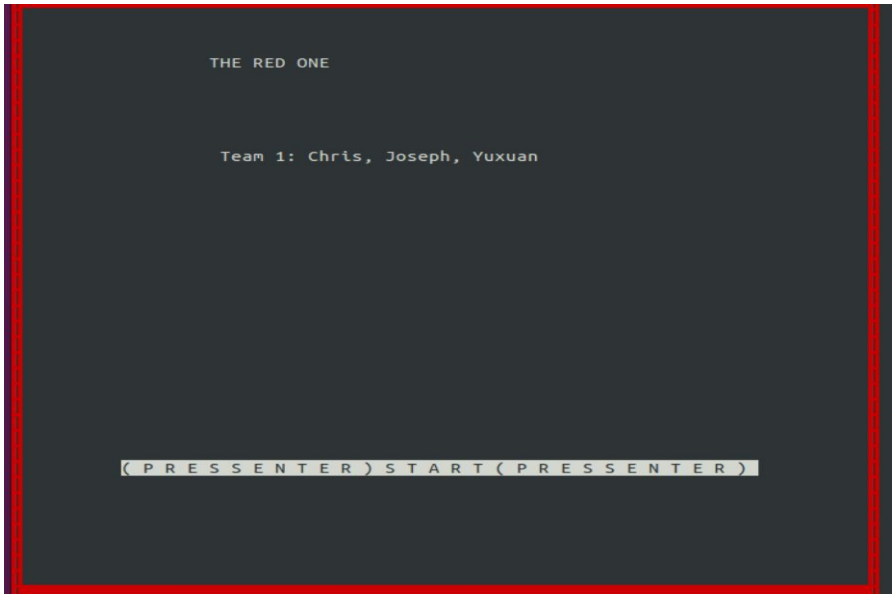
lyx0203- finished scorekeeping using file io

cpadilla: commented and cleaned up code ready to be delivered

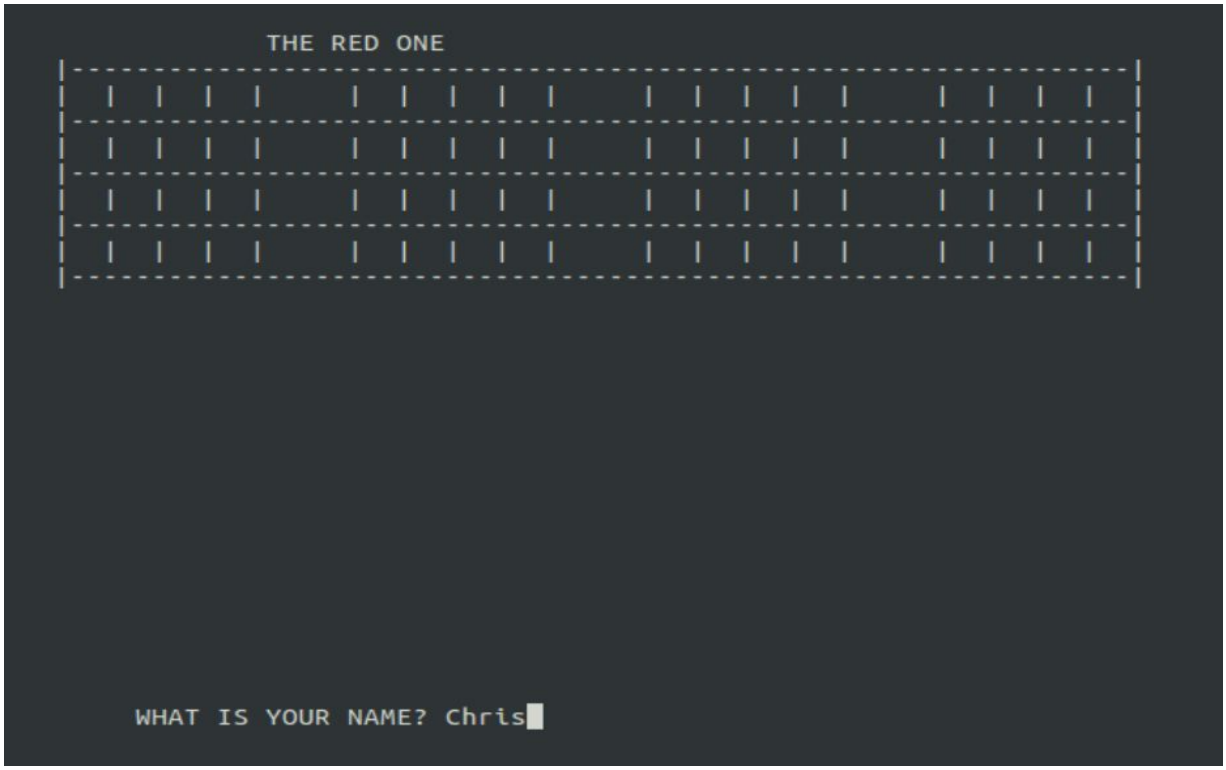
Test Sessions



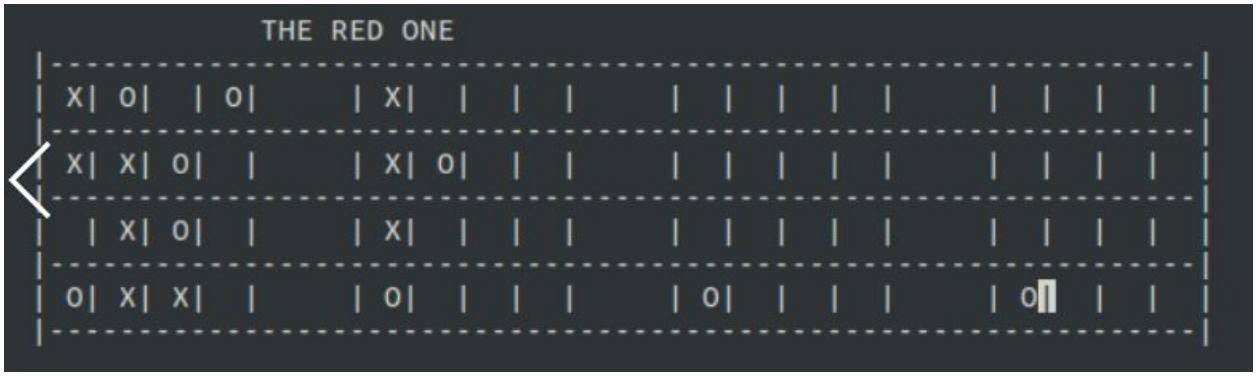
The first of the major test sessions was to ensure the agent the user is playing is playing optimally (to the best of its ability). This non-user-friendly version of the program was used by Christopher to develop, run and test the AI fully.

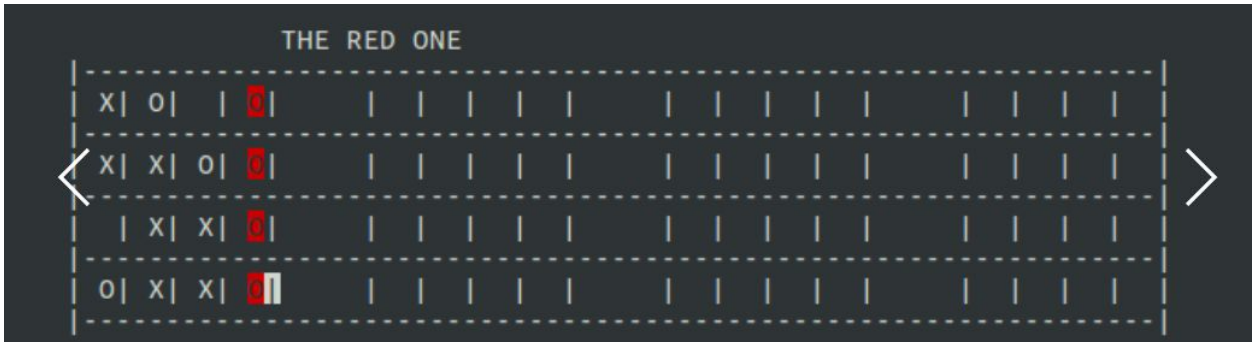


The splash screen. Pressing enter got the game to start.



The initial board. All testing for the board's position was done internally by Joseph to ensure the program was seeing the spaces as the correct value.





Gameplay testing was done by selecting the number of levels the tree would be and ensuring it acted optimally to its current configuration. We also ensured there

was blinking when the game was won (either by the user or computer, but mostly the computer).

Listing of Programs

Node class(Used for minmax tree):

```cpp
#ifndef NODE_H
#define NODE_H

#include <iostream>

struct Node {
    static std::vector<Node*> bunch_nodes;
    Node* parent;
    int value;
    std::vector<int> game_state;
    std::vector<Node*> children;
    bool max;
    bool terminal_node;

    Node(Node* parent, std::vector<int> game_state, bool max, bool terminal_node):
        parent(parent), game_state(game_state), max(max),
        terminal_node(terminal_node) {
            add();
        }
    void add() {
        bunch_nodes.push_back(this);
    }
};

#endif
```

Logic class(Used for AI movement):

```cpp
#ifndef LOGIC_H
#define LOGIC_H

#include <vector>
#include "Node.h"
#include "Tree.h"

#include <vector>
#include "Node.h"
#include "Tree.h"

/* This is where all the AI logistic functions are stored */

int spots_filled = 0;
std::vector<std::vector<int>> columns;
std::vector<Node*> Node::bunch_nodes;

/* Returns true if the inputted number is in a row specified by its
    upper and lower bound */
bool row_value(int num, int lower_bound, int upper_bound) {;
    for (int i = 0; i < 4; i++) {
        if (num >= lower_bound && num <= upper_bound) {
            return true;
        }
        lower_bound+=16;
        upper_bound+=16;
    }
    return false;
}

/* Returns true if the inputted number is in a column specified by its
    upper and lower bound */
bool column_value(int num, int lower_bound, int upper_bound) {
    for (int i = 0; i < 4; i++) {
        if (num >= lower_bound && num <= upper_bound) {
            return true;
        }
        lower_bound+=4;
        upper_bound+=4;
    }
}

/* Generates an internal board used to convert between vector
    and numeric notation */
void generate_helper_board() {
    int num1 = 1, num2 = 5, num3 = 9, num4 = 13;
    for (int j = 0; j < 4; j++) {
        for (int i = 0; i < 4; i++) {
            std::vector<int> new_vector;
```

```
            std::vector<int> new_vector;
            new_vector.resize(4);
            new_vector.at(0) = num1 + i + j * 16;
            new_vector.at(1) = num2 + i + j * 16;
            new_vector.at(2) = num3 + i + j * 16;
            new_vector.at(3) = num4 + i + j * 16;
            columns.push_back(new_vector);
        }
    }
}

/* Finds in vector notation the numerical position of interest
    ex. converts 16 to (0, 2) */
std::vector<int> convert_to_vector(int num) {
    std::vector<int> vector_notation;
    vector_notation.resize(2);

    if (row_value(num, 1, 4)) vector_notation.at(0) = 0;
    else if (row_value(num, 5, 8)) vector_notation.at(0) = 1;
    else if (row_value(num, 9, 12)) vector_notation.at(0) = 2;
    else vector_notation.at(0) = 3;

    for (int i = 0; i < columns.size(); i++) {
        for (int j = 0; j < 4; j++) {
            if (num == columns.at(i).at(j)) {
                vector_notation.at(1) = i; break;
            }
        }
    }

    return vector_notation;
}

/* Returns in vector notation the set of points that are the winning coordinates
    (you should only ever call this function after the game is won) */
std::vector<std::vector<int>> find_winning_combo(Tree &minimax_tree) {
    std::vector<std::vector<int>> return_vector;
    std::vector<int> winning_vector = minimax_tree.find_winning_combo(minimax_tree.check_win().at(1));
    for (int i = 0; i < winning_vector.size(); i++) {
        return_vector.push_back(convert_to_vector(winning_vector.at(i)));
    }

    return return_vector;
}

/* Finds numerical position of interest given its vector notation
    ex. converts (0, 2) to 16 */
int convert_to_num(int x, int y) {
    return columns.at(y).at(x);
```

```
    (for gorey details see Node.h/Node.cpp and Tree.h/Tree.cpp */
int AI_Move(Tree &minimax_tree, int difficulty) {
    /* initializes minimax tree with a specified depth;
       larger depth means smarter but slower program */
    minimax_tree.initialize(difficulty);

    /* after all nodes are generated we compute the minimax for
       all nodes to find the top node's value */
    minimax_tree.evaluate_nodes();

    /* the AI will take the path that led to the top node's current value */
    return minimax_tree.decide_next_move();
}

/* The ncurses logic calls this middleman to know generate AI move and know
    where to put the AI's X or O */
std::vector<int> send_robot_move(Tree &minimax_tree, bool player_letter, int difficulty)
    int ai_move = AI_Move(minimax_tree, difficulty);
    minimax_tree.current_game_state.at(ai_move - 1) = !player_letter;
    return convert_to_vector(ai_move);
}

/* The ncurses logic calls this middleman so the AI knows what position
    the human has picked */
void receive_player_move(int row, int col, Tree &minimax_tree, bool player_letter) {
    int move = convert_to_num(row, col);
    minimax_tree.current_game_state.at(move - 1) = player_letter;
}

#endif

```

Tree class (used for actual minmax tree):

```h
#ifndef TREE
#define TREE

#include <iostream>
#include <vector>
#include "Node.h"

class Tree {
    Node* _root;
public:
    /* The game-state for minimax purposes is stored as a
       one-dimensional array with 64 elements */
    std::vector<int> current_game_state;

    /* The win_states are generated every time the program starts,
       used to determine if someone (anyone) has won the game */
    std::vector<std::vector<int>> win_states;

    bool _player_x;

    /* Constructor will take in whether the player is X or O (true and false)
       and initializes the internal data where to board state is stored */
    Tree(bool x): _player_x(x) {
        current_game_state.resize(64);
        for (int i = 0; i < 64; i++) {
            current_game_state.at(i) = -1;
        }
    }

    void initialize(int depth);

    void build_tree(Node* parent, int lookaheads);

    void evaluate_nodes();

    int compute_minimax(Node* to_compute, int alpha, int beta);

    int decide_next_move();

    std::vector<bool> check_win();

    std::vector<int> find_winning_combo(bool who_won);

    void destroy_tree();

    void generate_win_states();

    void print_win_states();

    std::vector<std::vector<int>> possible_moves(std::vector<int> game_state, bool max);

    int util_eval(std::vector<int>& game_state);

    ~Tree();
};

#endif
```

```cpp
#include "Tree.h"

using namespace std;

void Tree::initialize(int depth) {
    _root = new Node(nullptr, current_game_state, _player_x, false);
    build_tree(_root, depth);
}
void Tree::build_tree(Node* parent, int lookaheads) {
    if (lookaheads == 0) return;
    vector<vector<int>> current_moves = possible_moves(parent->game_state, parent->max);
    for (int i = 0; i < current_moves.size(); i++) {
        parent->children.push_back(new Node(parent, current_moves.at(i), !parent->max, lookaheads == 1)
        build_tree(parent->children.at(i), lookaheads - 1);
    }
}

void Tree::evaluate_nodes() {
    compute_minimax(_root, -99999, 99999);
}

int Tree::compute_minimax(Node* to_compute, int alpha, int beta) {
    if (to_compute->terminal_node) /* terminal node */ {
        to_compute->value = util_eval(to_compute->game_state);
        return util_eval(to_compute->game_state);
    }
    else if (to_compute->max) /* maximizing node */ {
        int value = -99999;
        for (int i = 0; i < to_compute->children.size(); i++) {
            value = max(value, compute_minimax(to_compute->children.at(i), alpha, beta));
            alpha = max(alpha, value);
            if (beta <= alpha) break;
        }
        to_compute->value = value;
        return value;
    }
    else /* minimizing node */ {
        int value = 99999;
        for (int i = 0; i < to_compute->children.size(); i++) {
            value = min(value, compute_minimax(to_compute->children.at(i), alpha, beta));
            beta = min(beta, value);
            if (beta <= alpha) break;
        }
        to_compute->value = value;
        return value;
    }
}

int Tree::decide_next_move() {
```

```
    for (int i = 0; i < _root->children.size(); i++) {
        if (_root->children.at(i)->value == _root->value) {
            return _root->children.at(i)->game_state.at(64);
        }
    }
}

void Tree::generate_win_states() {
    /* Horizontal plane lines 1 to 16*/
    for (int i = 0; i < 16; i++) {
        vector<int> win_state;
        for (int j = 0; j < 4; j++) {
            win_state.push_back((j + 1) + 4 * i);
        }
        win_states.push_back(win_state);
    }

    /* Vertical plane lines 1 to 16 */
    for (int i = 0; i < 16; i++) {
        vector<int> win_state;
        for (int j = 0; j < 4; j++) {
            if (i < 4)
                win_state.push_back(i + 1 + 4 * j);
            else if (i < 8)
                win_state.push_back(i + 13 + 4 * j);
            else if (i < 12)
                win_state.push_back(i + 25 + 4 * j);
            else if (i < 16)
                win_state.push_back(i + 37 + 4 * j);
        }
        win_states.push_back(win_state);
    }

    /* Plane diagonals 1 to 4 */
    for (int i = 0; i < 4; i++) {
        vector<int> win_state;
        for (int j = 0; j < 4; j++) {
            win_state.push_back((j + 1) + (4 * j) + (i * 16));
        }
        win_states.push_back(win_state);
    }

    /* Plane diagonals 5 to 8 */
    for (int i = 0; i < 4; i++) {
        vector<int> win_state;
        for (int j = 0; j < 4; j++) {
            win_state.push_back((j + 4) + (2 * j) + (i * 16));
        }
        win_states.push_back(win_state);
```

```
    }

    /* Top-to-bottom vertical lines 1 to 16 */
    for (int i = 0; i < 16; i++) {
        vector<int> win_state;
        for (int j = 0; j < 4; j++) {
            win_state.push_back((i + 1) + (j * 16));
        }
        win_states.push_back(win_state);
    }

    /* Horizontal plane diagonals 1 to 4 */
    for (int i = 0; i < 4; i++) {
        vector<int> win_state;
        for (int j = 0; j < 4; j++) {
            win_state.push_back((i * 4 + 1) + (17 * j));
        }
        win_states.push_back(win_state);
    }

    /* Horizontal plane diagonals 5 to 8 */
    for (int i = 0; i < 4; i++) {
        vector<int> win_state;
        for (int j = 0; j < 4; j++) {
            win_state.push_back(4 * (i + 1) + (15 * j));
        }
        win_states.push_back(win_state);
    }

    /* Vertical plane diagonals 1 to 4 */
    for (int i = 0; i < 4; i++) {
        vector<int> win_state;
        for (int j = 0; j < 4; j++) {
            win_state.push_back((i + 1) + 20 * j);
        }
        win_states.push_back(win_state);
    }

    /* Vertical plane diagonals 5 to 8 */
    for (int i = 0; i < 4; i++) {
        vector<int> win_state;
        for (int j = 0; j < 4; j++) {
            win_state.push_back((i + 13) + 12 * j);
        }
        win_states.push_back(win_state);
    }

    /* Corner-to-corner diagonals */
    win_states.push_back({1, 22, 43, 64});
```

```
147     win_states.push_back({1, 22, 43, 64});
148     win_states.push_back({16, 27, 38, 49});
149     win_states.push_back({4, 23, 42, 61});
150     win_states.push_back({13, 26, 39, 52});
151 
152 }
153 
154 
155 /* Generates all possible moves for X or O from a current game state
156     returns moves as a vector of game states */
157 vector<vector<int>> Tree::possible_moves(vector<int> game_state, bool max) {
158     vector<vector<int>> move_vector;
159     game_state.resize(65);
160     if (max) /* maximizing player */ {
161         for (int i = 0; i < 64; i++) {
162             if (game_state.at(i) == -1) {
163                 game_state.at(i) = 1;
164                 game_state.at(64) = i + 1;
165                 move_vector.push_back(game_state);
166                 game_state.at(i) = -1;
167             }
168         }
169     }
170 
171     else /* minimizing player */ {
172         for (int i = 0; i < 64; i++) {
173             if (game_state.at(i) == -1) {
174                 game_state.at(i) = 0;
175                 game_state.at(64) = i + 1;
176                 move_vector.push_back(game_state);
177                 game_state.at(i) = -1;
178             }
179         }
180     }
181 
182     return move_vector;
183 }
184 
185 /* Evaluates the utility function of any given node, only used to find
186     the utility of terminal nodes */
187 int Tree::util_eval(vector<int>& game_state) {
188     int util_func = 0;
189     for (int i = 0; i < 76; i++) {
190         int x_count = 0;
191         int o_count = 0;
192         for (int j = 0; j < 4; j++) {
193             if (game_state.at(win_states.at(i).at(j) - 1) == 0) {
194                 o_count++;
```

```
            }
            else if (game_state.at(win_states.at(i).at(j) - 1) == 1) {
                x_count++;
            }
        }
        if ((x_count > 0 && o_count > 0) || (x_count == 0 && o_count == 0)) continue;
        bool x = (x_count > 0);
        if (x) {
            if (x_count == 4) util_func += 1000;
            else if (x_count == 3) util_func += 100;
            else if (x_count == 2) util_func += 10;
            else util_func++;
        }
        else {
            if (o_count == 4) util_func -= 1000;
            else if (o_count == 3) util_func -= 100;
            else if (o_count == 2) util_func -= 10;
            else util_func--;
        }
    }
    return util_func;
}

vector<bool> Tree::check_win() {
    /* Returns true if a player has won the game,
       will also return a second true/false if X/O has won*/
    vector<bool> win_vec;
    win_vec.resize(2);
    win_vec.at(0) = false; win_vec.at(1) = false;
    for (int i = 0; i < 76; i++) {
        int x_count = 0;
        int o_count = 0;
        for (int j = 0; j < 4; j++) {
            if (current_game_state.at(win_states.at(i).at(j) - 1) == 1)
                x_count++;
            else if (current_game_state.at(win_states.at(i).at(j) - 1) == 0)
                o_count++;
        }
        if (x_count == 4) {
            win_vec.at(0) = true;
            win_vec.at(1) = true;
            break;
        }
        else if (o_count == 4) {
            win_vec.at(0) = true;
            break;
        }
    }
```

```
242     }
243     return win_vec;
244 }
245
246 vector<int> Tree::find_winning_combo(bool who_won) {
247     vector<int> return_vector;
248     int win_space;
249     for (int i = 0; i < 76; i++) {
250         int count = 0;
251         for (int j = 0; j < 4; j++) {
252             if (current_game_state.at(win_states.at(i).at(j) - 1) == who_won)
253                 count++;
254         }
255         if (count == 4) {
256             for (int j = 0; j < win_states.at(0).size(); j++) {
257                 return_vector.push_back(win_states.at(i).at(j));
258             }
259         }
260     }
261     return return_vector;
262 }
263
264 void Tree::destroy_tree() {
265     for (size_t i = 0; i < Node::bunch_nodes.size(); i++) {
266         delete Node::bunch_nodes.at(i);
267     }
268 }
269
270 void Tree::print_win_states() {
271     /* this function is currently being used for debugging purposes */
272     int k = 1;
273     for (int i = 0; i < win_states.size(); i++) {
274         cout << k << ": ";
275         for (int j = 0; j < win_states.at(i).size(); j++) {
276             cout  << win_states.at(i).at(j) << " ";
277         }
278         k++;
279         cout << endl;
280     }
281 }
282
283 Tree::~Tree() {
284     destroy_tree();
285 }
286
```

Bucket.cpp (Where we kept our graphics and main):

bucket.cpp ● HSKNNGrapphic.py scores.txt logic.h

```cpp
#include <ncurses.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <sstream>
#include <vector>
#include <string>
#include <fstream>
#include <chrono>
#include <thread>
#include <unistd.h>
#include "logic.h"
#include <algorithm>
#include <list>

#define YBOARDSTART 5
#define XBOARDSTART 5

std::vector< std::string > sayings = {
    "Fine. I'll gut you standing.",
    "Time to die.",
    "Not going down.",
    "Man, you're ugly.",
    "I'll tell you about my mother...",
    "Goina surrender?",
    "BANG BANG BANG - K R S 1",
    "Wa da da dang",
    "Ain't going out like that",
    "*blinks*",
    "DO YOU READ",
    "Bet you could suck a golfball through a gardenhose",
    "What a nancy",
    "Yeah hang on",
    "M u S t B e U n D e R 1 0 s E c O n D s"
};
std::string player_name = "";

// - - - - - - - - - I N I T - - - - - - - - -
void init_curses() {
    initscr();
    cbreak(); //NOT SURE IF NEEDED
    noecho(); //NOT SURE IF NEEDED
    keypad(stdscr, TRUE);
}

// - - - - - - - - - U S E R I N P U T - - - - - - - - -
//CALLED AT BEGINING OF THE GAME
std::string get_initials() {
    mvprintw(30, 10, "WHAT IS YOUR NAME? ");
```

```
    refresh();
    nocbreak();
    echo();
    std::string input;
    int ch = getch();
    int counter = 0;
    while(ch != '\n' && counter < 20 && ch != ' ') {
        counter++;
        input.push_back(ch);
        ch = getch();
    }
    cbreak();
    noecho();
    mvprintw(30, 10, "                                                ");
    std::ofstream output_file;
    output_file.open("scores.txt",std::ios_base::app);
    output_file << "\n" << input << " 0";
    player_name = input;
    return input;
}

// - - - - - - - - - U S E R I N P U T - - - - - - - - -
char get_x_or_o() {
    mvprintw(30, 10, "PRESS X TO START FIRST OR O TO START SECOND");
    refresh();
    int ch = getch();
    if(ch == 'x' || ch == 'X'){
        mvprintw(30, 10, "                                              ");
        return 'X';
    }
    else if(ch == 'o' || ch == 'O'){
        mvprintw(30, 10, "                                              ");
        return 'O';
    }
    else{ return get_x_or_o(); }
}

// - - - - - - - - - U S E R I N P U T - - - - - - - - -
int how_hard() {
    mvprintw(31, 10, "HOW DIFFICULT DO YOU WANT THIS?(1-4)");
    refresh();
    int ch = getch();
    if(ch == '1'){
        mvprintw(31, 10, "I'm too young to die               ");
        return 1;
    }
    else if(ch == '2') {
        mvprintw(31, 10, "Hey, not too rough                 ");
        return 2;
```

```
    }
    else if(ch == '3') {
        mvprintw(31, 10, "Hurt me plenty                      ");
        return 3;
    }
    else if(ch == '4') {
        mvprintw(31, 10, "Nightmare!                          ");
        return 4;
    }
    else{ return how_hard(); }

}

// - - - - - - - - - - V E C T O R M A N I P U L A T I O N - - - - - - - - - -
std::vector< std::string > merge_scores(std::vector< std::string > input) {
    std::vector< std:: string > scores = input;
    for(int i = 0; i < scores.size(); i++) {
        int num_score = stoi(scores[i].substr(scores[i].find(" ")+1));
        std:: string name = scores[i].substr(0, scores[i].find(" "));
        for(int j = i + 1; j < scores.size(); j++){
            int other_score = stoi(scores[i].substr(scores[i].find(" ")+1));
            std::string other_name = scores[j].substr(0, scores[j].find(" "));
            if(name == other_name){
                num_score += other_score;
                scores[i] = name + " " + std::to_string(num_score);
                scores.erase(scores.begin() + j);
            }
        }
    }
    return scores;
}

// - - - - - - - - - - V E C T O R M A N I P U L A T I O N - - - - - - - - - -
std::vector<std::string> sort_scores(std::vector< std::string > highscores){
    std::list<std::pair< int, std::string> > list_pair;
    std::vector<std::string> sorted_highscores;
    for(int i=0; i<highscores.size(); i++){
        std::string line = highscores[i];
        for(int j=0; j<line.length(); j++){
            char ch = line[j];
            if(isdigit(ch)){
                int num = stoi(line.substr(j));
                list_pair.push_back(make_pair(num, highscores[i]));
                break;
            }
        }
    }
    list_pair.sort();
    std::vector<std::pair<int, std::string>> v;
```

```
148     v.reserve(list_pair.size());
149     std::copy(std::begin(list_pair), std::end(list_pair), std::back_inserter(v));
150     for(int i=0; i<v.size(); i++){
151         std::pair<int, std::string> p1 = v[i];
152         sorted_highscores.push_back(std::get<1>(p1));
153     }
154     std::reverse(sorted_highscores.begin(), sorted_highscores.end());
155     return merge_scores(sorted_highscores);
156 }
157 
158 // - - - - - - - - - - R E A D F I L E - - - - - - - - - -
159 std::vector< std::string > read_scores_from_file(std::string score_file_name){
160     std::ifstream file(score_file_name.c_str());
161     std::vector< std::string > highscores;
162     int counter = 1;
163     std::string line;
164     while(std::getline(file, line)) {
165         highscores.push_back(line);
166     }
167     return sort_scores(highscores);
168 }
169 
170 // - - - - - - - - - - V E C T O R M A N I P U L A T I O N - - - - - - - - - -
171 std::vector< std::string > increment_score(std::vector< std::string > input) {
172     std::vector< std::string > scores = input;
173     for(int i = 0; i < scores.size(); i++) {
174         int num_score = stoi(scores[i].substr(scores[i].find(" ")+1));
175         std:: string name = scores[i].substr(0, scores[i].find(" "));
176         if(name == player_name) {
177 
178             num_score++;
179             std::string new_name_and_score = name + " " + std::to_string(num_score);
180             scores[i] = new_name_and_score;
181         }
182     }
183     return scores;
184 }
185 
186 // - - - - - - - - - - W R I T E F I L E - - - - - - - - - -
187 void write_scores_to_file(std::vector< std::string > scores, bool did_they_win){
188     std::ofstream file("scores.txt", std::ios::in);
189     file.close();
190     file.open("scores.txt", std::ios::out | std::ios::trunc);
191 
192     file.close();       //CLEARS THE FILE OUT!
193     std::ofstream file_("scores.txt");
194     if(did_they_win) {
195         scores = increment_score(scores);
196         scores = sort_scores(scores);
```

```
    }
    int size = scores.size();
    if(scores.size() > 5) {
        size = 5;
    }
    for(int name = 0; name < size; name++) {
        file_ << scores[name] << "\n";
    }

}

// - - - - - - - - - - U S E R I N P U T - - - - - - - - - -
bool play_again_prompt(){
    mvprintw(30, 10, "                                                  ");
    mvprintw(30, 10, "PLAY AGAIN? (y/n) ");
    refresh();
    int ch = getch();
    if(ch == 'y'){
        return TRUE;
    }
    else if(ch == 'n'){
        mvprintw(32, 15, "Later gator!");
        mvprintw(33, 15, "Press any key to exit.");
        return FALSE;
    }
    else{
        return play_again_prompt();
    }
}

// - - - - - - - - - - D R A W - - - - - - - - - -
//CALLED AT BEGINING OF THE GAME
void draw_start_button() {
    mvprintw(4, 18, "THE RED ONE");
    mvprintw(10, 19, "Team 1: Chris, Joseph, Yuxuan");
    attron(A_REVERSE);
    mvprintw(30, 10, "( P R E S S E N T E R ) S T A R T ( P R E S S E N T E R )");
    attroff(A_REVERSE);
    refresh();
    char enter_check;
    while(enter_check != '\n'){
        enter_check = getch();
    }
    mvprintw(10, 19, "                              ");
    mvprintw(30, 10, "                                                  ");
}
// - - - - - - - - - - D R A W - - - - - - - - - -
//DRAWS THE RED OUTLINE
void draw_splash() {
```

```
246     start_color();
247     init_pair(1, COLOR_BLACK, COLOR_RED);
248     attron(COLOR_PAIR(1));
249     for(int col = 1; col < 78; col++) {
250         mvaddch(0, col,'-');
251         mvaddch(38, col, '_');
252     }
253     for(int row = 0; row < 39; row++) {
254         mvaddch(row,0,'|');
255         mvaddch(row,78,'|');
256     }
257     attroff(COLOR_PAIR(1));
258 }
259 
260 // - - - - - - - - - - D R A W - - - - - - - - - -
261 //DRAWS ERASE INITIAL GAME BOARD
262 void clear_board() {
263     mvprintw(15+YBOARDSTART,17,"DEMO OVER! PLEASE LIKE AND SUBSCRIBE!");
264     mvprintw(15+YBOARDSTART,17,"                                      ");
265     for(int row = 0; row < 9; row++) {
266         for(int col = 0; col < 69; col++) {
267             mvaddch(YBOARDSTART + row, XBOARDSTART + col, ' ');
268         }
269     }
270 }
271 
272 // - - - - - - - - - - D R A W - - - - - - - - - -
273 //DRAWS ERASE INITIAL GAME BOARD
274 void draw_instructions() {
275     mvprintw(35,15,"PRESS SPACE TO PLACE A MARK! ARROW KEYS TO MOVE!");
276 }
277 
278 // - - - - - - - - - - D R A W - - - - - - - - - -
279 //DRAWS INITIAL BOARD
280 void draw_board() {
281     start_color();
282     init_pair(2, COLOR_WHITE, COLOR_BLACK);
283     attron(COLOR_PAIR(2));
284     for(int row = 0; row < 9; row++) {
285         int board_separator = 0;
286         for(int col = 0; col < 69; col += 3) {  //CAUTION ITERATOR +3
287             if(board_separator != 5) {      //IF ON A BOARD
288                 mvaddch(YBOARDSTART + row, XBOARDSTART + col, '|');
289                 board_separator++;
290             }
291             else {
292                 board_separator = 0;
293             }
294         }
```

```
295         if((row % 2) == 0) {
296             for(int col = 1; col < 66; col++){
297                 mvaddch(YBOARDSTART + row, XBOARDSTART + col, '-');
298             }
299         }
300     }
301     refresh();
302 }
303
304 // - - - - - - - - - - D R A W - - - - - - - - - -
305 //DRAWS HIGHSCORES
306 void draw_scores(std::vector< std::string > ledger) {
307     int size = ledger.size();
308     if(ledger.size() > 5){
309         size = 5;
310         std::string name = ledger[ledger.size()-1];
311         mvprintw(10 + 5 + YBOARDSTART, 1, "                              " );
312         mvprintw(10 + 5 + YBOARDSTART, 1, name.c_str());
313     }
314     for(int name_index = 0; name_index < size; name_index++){
315         std::string name = ledger[name_index];
316         mvprintw(10 + name_index + YBOARDSTART, 1, "                                  " );
317         mvprintw(10 + name_index + YBOARDSTART, 1, name.c_str());
318     }
319 }
320
321 // - - - - - - - - - - M A K E - - - - - - - - - -
322 //INITIAL CALL TO CREATE EMPTY DATA STRUCTURE
323 std::vector< std::vector< char > > make_board() {
324
325     std::vector< std::vector< char > > * board =
326             new std::vector< std::vector< char > >;
327
328     for(int row = 0; row < 4; row++ ) {
329         std::vector< char > temp_row;
330         for(int col = 0; col < 16; col++){
331             temp_row.push_back(' ');
332         }
333         board->push_back(temp_row);
334     }
335     return *board;
336 }
337 // - - - - - - - - - - C A L C U L A T I O N - - - - - - - - - -
338 int y_pos_to_row(int y_pos) {
339     return (y_pos - 1 - YBOARDSTART) / 2;          //WEIRD CONVERSION FROM ncurses COORDS
340 }
341
342 // - - - - - - - - - - C A L C U L A T I O N - - - - - - - - - -
343 int x_pos_to_col(int x_pos) {
```

```
    int col = (x_pos - 2 - XBOARDSTART) / 3;
    if(col > 3) {
        col = col - 2;
        if(col > 9) {
            col = col -2;
            if(col > 13) {
                col = col - 2;
            }
        }
    }
    return col;
}

// - - - - - - - - - - C A L C U L A T I O N - - - - - - - - - -
int row_to_y_pos(int row) {
    return row * 2 + 1 + YBOARDSTART;
}

// - - - - - - - - - - C A L C U L A T I O N - - - - - - - - - -
int col_to_x_pos(int col) {
    int x_pos = col;
    if(col > 11){
        x_pos += 6; //6
    } else if(col > 7) {
        x_pos += 4; //4
    } else if(col > 3) {
        x_pos += 2; //2
    }
    return x_pos*3+2+XBOARDSTART;
}

// - - - - - - - - - - O P E N ? - - - - - - - - -
//CHECKS IF SPOT HAS BEEN PLAYED IN YET
bool spot_open(std::vector< std::vector< char > > board, int y, int x) {
    int row = y_pos_to_row(y);
    int col = x_pos_to_col(x);
    if(board[row][col] == ' ') {    //' ' IS THE EMPTY CHAR FOR THE STRUCTURE
        return TRUE;
    } else { return FALSE; }
}

// - - - - - - - - - - O P E N ? - - - - - - - - -
bool spot_valid(int y_pos, int x_pos) {
    int row = y_pos_to_row(y_pos);
    int col = (x_pos - 2 - XBOARDSTART) / 3;
    if ((col == 4) || (col == 5) || (col == 10) || (col == 11) || (col == 16) || (col == 17)){
        return FALSE;
    }
    else{ return TRUE; }
```

```
// - - - - - - - - - B O N U S - - - - - - - - -
void blink_red(std::vector< std::vector< int >> win_set, char marker) {
    start_color();
    init_pair(4, COLOR_BLACK, COLOR_RED);
    attron(COLOR_PAIR(4));
    init_pair(5, COLOR_WHITE, COLOR_BLACK);
    int user_input_char = 0;
    for(int i = 0; i < 5; i++){
        for(int i = 0; i < 4; i++) {
            attron(COLOR_PAIR(4));
            mvaddch(row_to_y_pos(win_set[i][0]), col_to_x_pos(win_set[i][1]), marker);
            refresh();
        }
        std::this_thread::sleep_for(std::chrono::milliseconds(500));
        for(int i = 0; i < 4; i++) {
            attron(COLOR_PAIR(5));
            mvaddch(row_to_y_pos(win_set[i][0]), col_to_x_pos(win_set[i][1]), marker);
            refresh();
        }
        std::this_thread::sleep_for(std::chrono::milliseconds(500));

    }
    attroff(COLOR_PAIR(5));
}


// - - - - - - - - - G A M E P L A Y - - - - - - - - -
bool play_game(char p_c, int difficulty) {
    bool win = FALSE;
    Node::bunch_nodes.clear();
    bool x_or_o = p_c != 'X';
    spots_filled = 0;
    Tree minimax_tree(x_or_o);
    minimax_tree.generate_win_states();
    std::vector< std::vector< char > > board = make_board();
    int y_pos = 1+YBOARDSTART, x_pos = 2+YBOARDSTART;
    int user_input_char = 0;    //BASE, IS OVERWRITTEN CONSTANTLY
    char player_char = p_c;
    char turn = 'X';
    move(y_pos,x_pos);          //CRUCIAL OTHERWISE CURSOR STARTS AT BOTTOM
    while(user_input_char != 'q'){
        if(player_char == turn)
        {
            user_input_char = getch();
            my_move(user_input_char, y_pos, x_pos); //SAFETY CHECKER MOVEMENT
            if((user_input_char == ' ')){
                getyx(stdscr, y_pos, x_pos);
                if(spot_open(board, y_pos, x_pos) && spot_valid(y_pos, x_pos)){
```

```
}

// - - - - - - - - - - U P D A T E - - - - - - - - - -
//ADDS player_char TO THE board FOR DATA KEEPING
void spot_update(std::vector< std::vector< char > > * board, int y, int x, char& player_char) {
    int row = y_pos_to_row(y);
    int col = x_pos_to_col(x);
    board->at(row).at(col) = player_char;
    if(player_char == 'O'){
        player_char = 'X';
    } else { player_char = 'O';}
}

// - - - - - - - - - - M O V E - - - - - - - - - -
//CHECKS SAFETY OF MOVEMENT AND MOVES CURSOR
void my_move(int user_input_char, int& y_pos, int& x_pos) {
    if(user_input_char != ' ') {
        switch (user_input_char) {
            case KEY_UP:
                if(y_pos != 1+YBOARDSTART) { move(y_pos = y_pos - 2, x_pos); }
                break;
            case KEY_DOWN:
                if(y_pos != 7+YBOARDSTART) { move(y_pos = y_pos + 2, x_pos); }
                break;
            case KEY_LEFT:
                if(x_pos != 2+XBOARDSTART) { move(y_pos, x_pos = x_pos - 3); }
                break;
            case KEY_RIGHT:
                if(x_pos != 65+XBOARDSTART) { move(y_pos, x_pos = x_pos + 3); }
                break;
        }
    }
    refresh();
}

// - - - - - - - - - - D E B U G - - - - - - - - - -
//FUNCTION TO MAKE SURE BOARD DATA IS BEING PRINTED CORRECTLY
void debug_send_to_file(std::vector< std::vector< char > > board){
    std::ofstream output_file("debug_info.txt");
    output_file << "-MOST RECENT TEST-";        //COMMENT OUT THIS LINE TO MAKE CSV
    for(int row = 0; row < 4; row++){
        output_file << "\n";
        for(int col = 0; col < 16; col++){
            output_file << board[row][col];
            if(col != 15) { output_file << " , "; }
        }
    }
}

```

```
                    mvaddch(y_pos, x_pos, player_char);
                    spot_update(&board, y_pos, x_pos, turn);
                    mvprintw(10+YBOARDSTART,30, "                                    ");
                    move(y_pos, x_pos);
                    std::this_thread::sleep_for(std::chrono::milliseconds(100));
                    receive_player_move(y_pos_to_row(y_pos),x_pos_to_col(x_pos), minimax_tree, x_or_o);
                    spots_filled++;
                    refresh();
                }
                else{ mvprintw(10+YBOARDSTART, 30, "BAD MOVE! YOU'LL GET IT NEXT TIME.."); move(y_pos,x_pos); }
            }
        } else {
            int num = rand() % sayings.length();
            mvprintw(20+YBOARDSTART, 10, sayings[num].c_str());
            refresh();
            std::vector<int> robot_moves = send_robot_move(minimax_tree, x_or_o, difficulty);
            mvprintw(20+YBOARDSTART, 10, "                                    ")
            int robot_x_move = col_to_x_pos(robot_moves[1]); //0 CHANGES TO WHAT AI MOVE RETURNS
            int robot_y_move = row_to_y_pos(robot_moves[0]);
            mvaddch(robot_y_move, robot_x_move, turn);
            spot_update(&board, robot_y_move, robot_x_move, turn);
            move(robot_y_move, robot_x_move);
            spots_filled++;
        }
        if(minimax_tree.check_win().at(0)) {
            if(minimax_tree.check_win().at(1) && x_or_o) {
                win = TRUE;
                mvprintw(15+YBOARDSTART,17,"YOU WON!");
            }
            if (!minimax_tree.check_win().at(1) && !x_or_o) {
                win = TRUE;
                mvprintw(15+YBOARDSTART,17,"YOU WON!");
            }
            std::vector<std::vector<int>> winner = find_winning_combo(minimax_tree);
            char win_char = !minimax_tree.check_win().at(1) ? 'X' : 'O';
            blink_red(winner, win_char);
            break;
        }
        if(spots_filled == 64){ break; }
    }
    return win;
}

int main() {
    generate_helper_board();
    init_curses();
    draw_splash();
    draw_start_button();
    int difficulty;
```

```
525                 blink_red(winner, win_char);
526                 break;
527             }
528             if(spots_filled == 64){ break; }
529         }
530         return win;
531     }
532
533     int main() {
534         generate_helper_board();
535         init_curses();
536         draw_splash();
537         draw_start_button();
538         int difficulty;
539         bool play = TRUE;
540         while(play){
541             clear_board();
542             draw_board();
543             get_initials();
544             draw_scores(sort_scores(read_scores_from_file("scores.txt")));
545             draw_instructions();
546             difficulty = how_hard();
547             char option = 'O';
548             option = get_x_or_o();
549             bool win = play_game(option, difficulty);
550             write_scores_to_file(read_scores_from_file("scores.txt"), win);
551             play = play_again_prompt();
552         }
553         getch();
554         endwin();
555     }
```