Pancake Sort (Artificial Intelligence)

CSCE 315 – 508, Project 3

Team 13:
Stanley Chen
Andrew Chin
Jingchun Qu
Alexander Staggs[1]
David Tieding

[1] Denotes team leader

Section 1 - Team Information

Team 13 - Jingchun Qu, Alexander Staggs, Andrew Chin, David Tieding, Zicheng Chen

Team Member Contributions

- Jingchun Qu (20%) - AI Player implementation, Pancake Data structure, Engine Implementation
- Alexander Staggs (20%) - Initial pancake order setting, front-end of user and AI move selections.
- Andrew Chin (20%) - Reading and Writing to File, Managing the Top 5 High Scores, Min Number of Moves Extra Credit, Give Me A Hint Extra Credit
- David Tieding (20%) - Pancake Display, Querying Users for Difficulty and Number of Pancakes
- Zicheng Chen (20%) - Splash Screen Display and Instructions, Game demonstration on the Splash Screen Extra Credit

Each team member was valuable and integral to the completion of this project. Each team member was very involved in the planning, design, coding, testing, and concluding phase. Work was completed in a timely manner and communication was used to keep every team member informed over the changing timeline of our project. As shown above, each team member contributed an equal amount of work to this project.

Section 2 - Problem Statement

Problem: We would like to understand how artificial intelligence fits into the concept of gaming and implementing NPCs (Non-playable characters). In addition to creating a single player game, we aim to understand how the minimax search algorithm can be used to create a smart virtual opponent that plays against the user.

Purpose: In this project, we are implementing a simple turn-based Pancake Sorting game. The purpose is to create a challenging puzzle game while at the same time introducing one of the fundamental search algorithms behind AI game search: minimax search. This search algorithm will be utilized to create a virtual opponent that the user will play against. The goal of the game is for the user to sort their stack of pancakes in as few moves as possible, preferably less than the virtual opponent. In gameplay, the virtual opponent should analyse its current state by performing a minimax search and use these results to select its next move.

Intended User: This project is intended for any user: developer or not. We introduce the minimax search in the form of a virtual opponent during gameplay. The gameplay should be fun and enjoyable, while providing some insight into emulating “intelligent behavior” in non-playable characters in gaming.

Section 3 - Restrictions and Limitations

The restrictions and limitations of the program are largely driven by the project specifications themselves. For instance the program is limited to only playing the pancake game, as opposed to multiple games or even a different game altogether. Another restriction that was driven by the project specifications was that the only acceptable language was c++. A time restriction was also initially added to the project. Four weeks was the maximum amount of time allowed to complete the program. The program must utilize ncurses as well.

Some other restrictions and limitations our program faces is the number of pancakes a user is allowed to ask for in a game. When the game begins the user is first shown a series of title screens and instructions for how to play the game before eventually being asked to specify the number of pancakes the user would like in their stack. This feature was regulated according to the project specifications. This means that a user may ask for a stack of pancakes that contains n number of pancakes, where n is an integer between one and nine inclusive. The user is then asked to input the level of difficulty they would like to face. This is also a regulated feature of the game. A user may only choose a difficulty between 1 and n, where n is the total number of pancakes in the stack.

Several limitations reside in the gameplay itself. One such limitation revolves around the inputs a user may give in order to manipulate the cursor. The user must use the ‘W’, ‘S’ and ‘Enter’ keys in order to move the cursor and select the pancake they wish to flip. However, even the act of flipping pancakes comes with restrictions. The user is not allowed to flip the first pancake in the stack, and the user must wait until the AI finishes making its moves before the user may make another move.

Naturally the AI program and functionality faces a few restrictions as well. Chief among these is the maximum size of the minmax tree that is implemented in our program. The minmax tree can only be of size n, where n is the number of pancakes in the stack. The AI faces a small, albeit potentially significant limitation in that it does not take into account the state of the user. This means that the AI can not regulate itself to match the user’s skill level more dynamically and appropriately.

Section 4 - Explanation of Approach

User Display Screens:

The initial screen displays the word “Pancake”, team name, and all the names. In addition to that, a simple example of the game displays on the top half of the screen while a prompt that says “Press enter to continue” on the bottom half. All the content displayed are adjusted to fit a 80*40 screen in the terminal on Macbook.

The screen requires functions from the ncurses library. The display of names are done by using “mvaddstr” function. The blinking prompt is coded using a new window that is 30 spaces in width and 3 spaces in height. The initial color is black for the prompt and the color changes every second. The extra credit part is coded using three states of the game. The initial stage (unsorted pancake) is first displayed. It switches to the blinking prompt every 500 ms. We did not choose to use 1 second as suggested because 500ms is the one we used for blinking through the entire program and we want to keep that consistent. The program reaches the final state after it loops for three times (visits the first and second state each three times).

Initially the user is given both instructions and options for the difficulty and number of pancakes the user would like. This feeds the pancake generation process by providing the functions with the appropriate number of pancakes and whether a random order should be generated or if a user order should be used instead.

The pancakes themselves are displayed using string vectors, where every line in the “pancake box” comprises an element in the vector. Each element in the vector(“pancake box”) is centered appropriately. This keeps the pancakes lined up on the screen. The pancakes utilize “pancake box” wrapper classes that allow us to display the pancakes in any order and in any position on the screen.

File I/O and Score Saving

The approach for this section was simple. Have an easy way to read and write information from a text file. Additionally, we need to make changes to the data and then update the file to reflect the changes.

The first thing that was needed was a defined format for reading and writing to a text file to store the top 5 high scores. We simply dedicate each line to a single player’s score and stats, which includes their 3 letter initials, the difficulty they played on , and their final game score.

Reading from a file is simple, each field is separated by a space, so using a simple loop with an input file stream does the job. Once we read the data from a text file, we store it in a vector of tuples. Each tuple contains 3 elements for each respective field. The vector is always contains 5 elements because we only care about the top 5 scores. When the player has configured their options and finished playing the game, we construct a new tuple object for the player. Then we perform a simple insertion sort over the locally stored vector of scores in order to determine if the user's score has a place in the top 5 scores. We then insert (or don't insert) the new tuple into the vector and update the file.

Just like reading, we simply use a output file stream to iterate through the array of tuples and write the contents in the same format we read them in.

Setting Initial Conditions:

The first piece of information needed from the user was whether stack assignment would be random or user-chosen. The user is initially prompted to choose random or user-assigned order selection. If random is selected, a random order is assigned using srand() and a time seed to make the result truly random. If a user order is selected, the user is prompted to enter the sizes of pancakes in the stack from top to bottom. These user inputs are checked to be correct inputs, being single digits, and an additional check prevents the user from entering duplicates.

Pancake Stack Representation/Implementation:

This pancake stacks are represented as a vector in the pancake stack class. It's implemented as a class to add functionality and simplify the code. The pancake class' main function is the flip function that takes an integer and makes the necessary adjustment.

User Selection Cursor

To the left of the printed pancake stacks, two columns are reserved for printing a little arrow (->) which points to a stack to move. Users can use 'w' and 's' to move the cursor to the desired location and 'enter' to make a move. It is not possible to make an invalid move, as attempts to move the cursor above the second pancake or below the bottom pancake are ignored.

User/AI move Blink

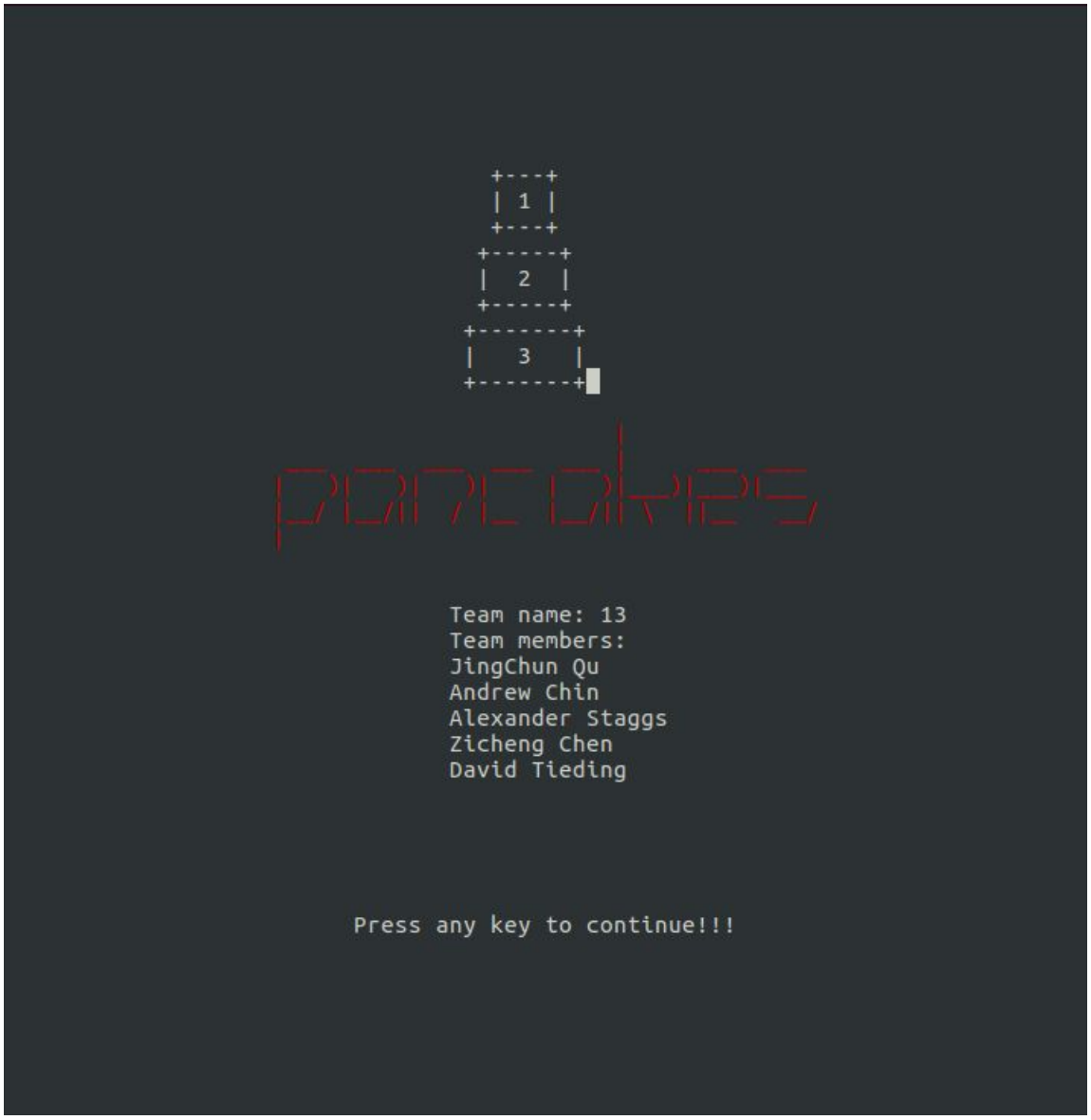
When either the AI or the user has selected their move, the portion of the stack that will be flipped will blink for 3 seconds. Blinking is performed by erasing the portion of the screen holding the sub-stack of pancakes, waiting half a second, then reprinting it, waiting half a second, and looping.

AI implementation:

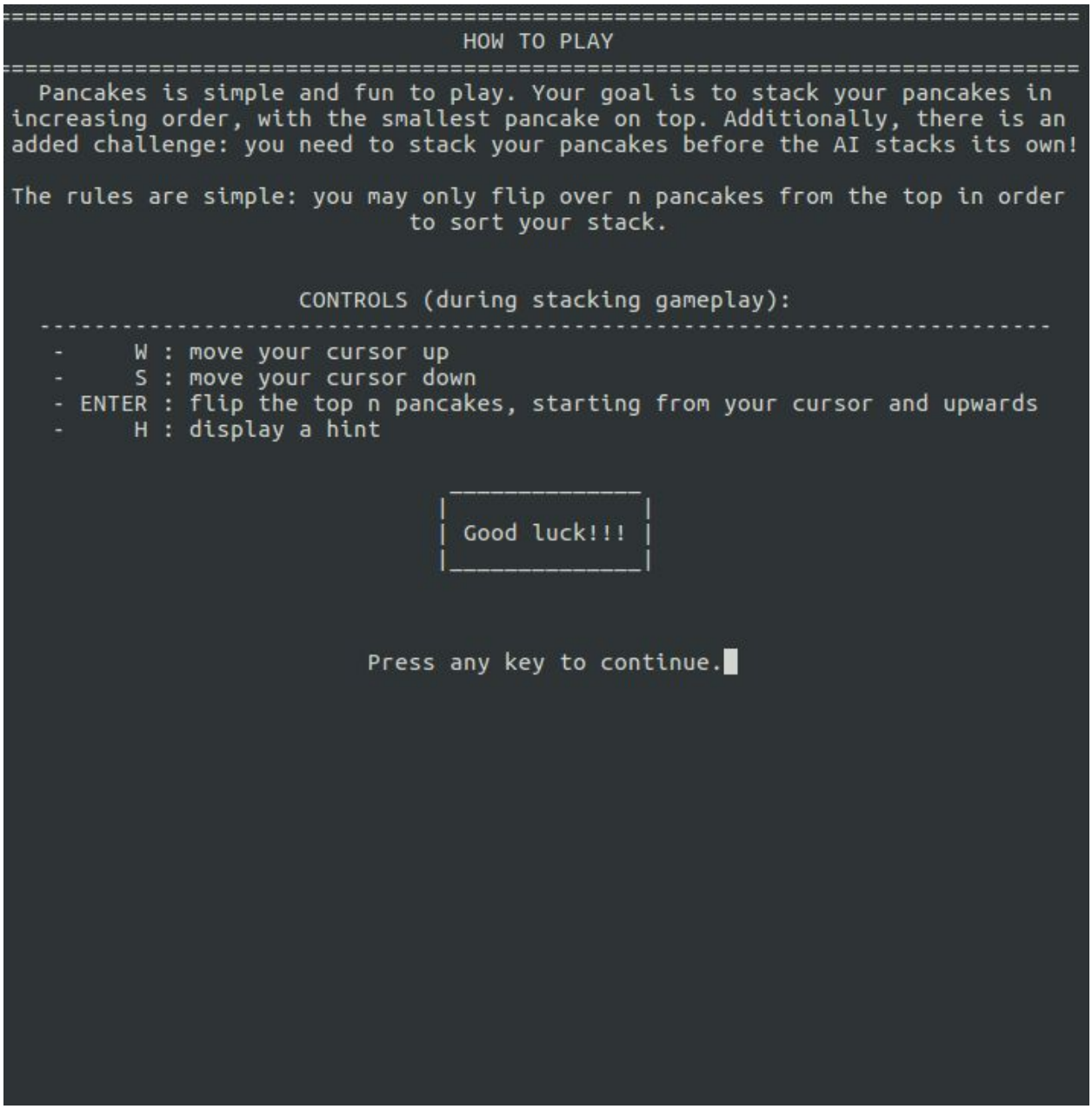
The algorithm that finds the best move for the AI takes a Pancake object and stores the state of the pancake stack into a node and constructs a tree of depth difficulty recursively. The recursive algorithm has base cases that would return when reaching a leaf node - a node that is at depth difficulty or is sorted. The leaf nodes are pushed into a vector in ai_tree class. The vector of nodes is then traversed to find the leaf node with the least reversals and minimal depth. The algorithm then starts from that node and goes up the tree to find the initial move for the current state that led to the current perceived optimal state. The reason for why we departed from the min-max algorithm was because it would only serve to complicate as an action on one stack does not alter the other player's progression with their stack in anyway. The reason why we decided against iterative deepening is because iterative deepening is depth first search algorithm with incremented depth which is good for searching as it saves memory but not so effective when we have to limit ourselves to a certain depth/difficulty and compare sortedness advantage (the nodes/states are not kept in iterative deepening). We also implemented ways to optimize this algorithm: if pancakes at the bottom are already sorted we would not make a node where the order of those bottom pancakes are changed.

Section 5 - Sample Run/Screenshots

i. The splash screen

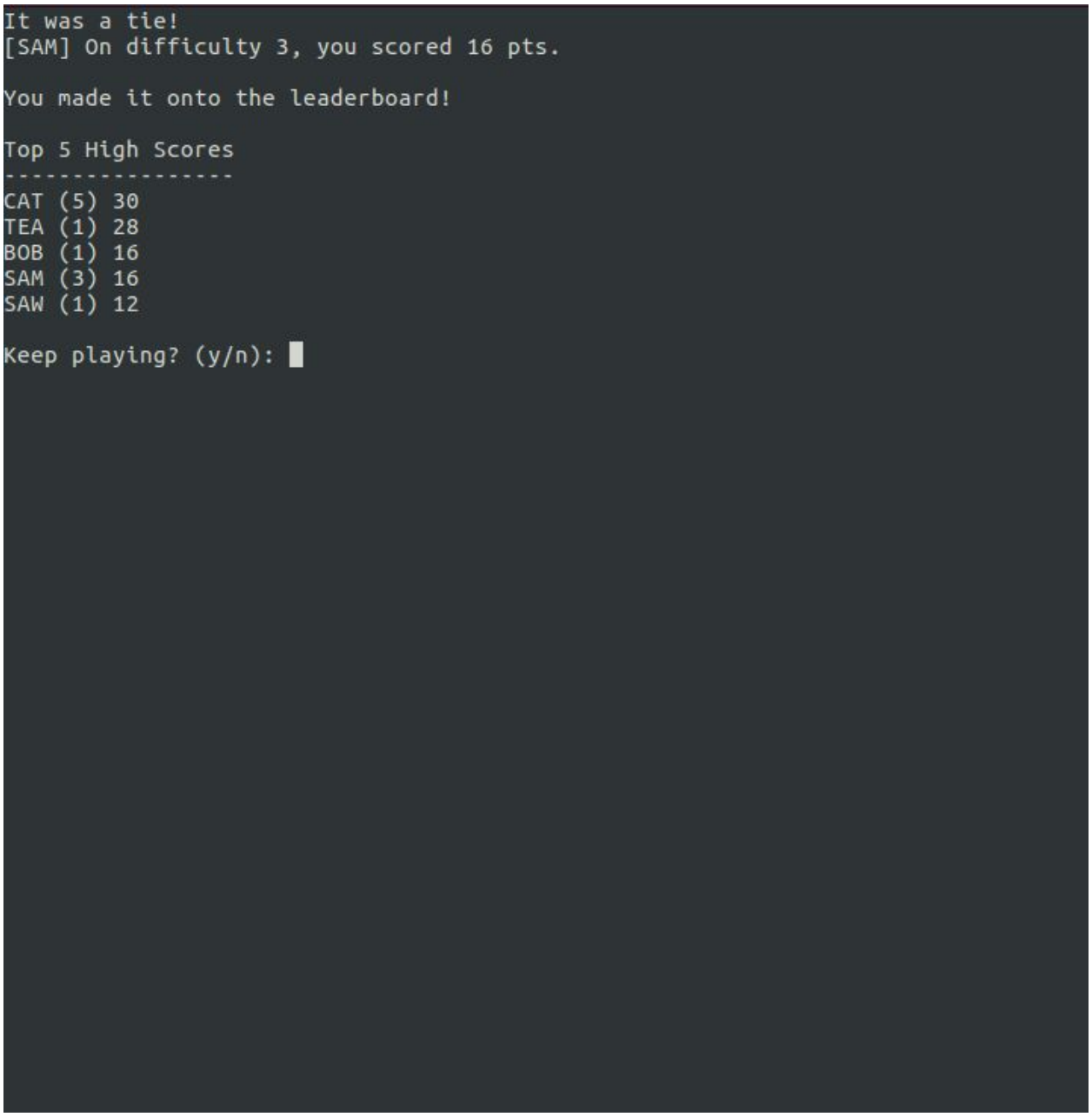


ii. The instructions screen



iii. A screen showing gameplay with pancake stacks of 4

iv. The results screen



v. A screen showing gameplay with pancake stacks of size 9

Section 6 - Results and Analysis

Splash Screen

The hard part of the splash screen is that I have to look up all the functions from the ncurses library. For the blinking prompt, I first tried to use an infinite for loop, but it did not work. I changed the blinking prompt to two states, one with white color and one with black color (invisible on the terminal window with black background). I make it switch between the two states and that finally worked.
For the simple example of the game, I tried to implement the functions from my teammate, but the final state kept showing the right edge of box_2, which I could not fix. The solution is to draw out three different states out and implement the similar method as the blinking prompt. It finally works correctly.

Setting Initial State

Initial state setting works correctly. Invalid user inputs prints a prompt to try again with correct values. Our initial design did not account for the inclusion of a random stack order option. This was remedied by including another screen to ask the user to choose between a random or user assigned stack order between the difficulty/number of pancake settings screen and the setting of the initial state screen. If the random option is selected, the process of asking the user for pancakes is skipped entirely.

File I/O and Score Saving

The results were fantastic. File operations worked just as they should, and new scores were added in correctly (i.e. low scores will not be added to the top 5 and higher scores would be added in). The approach stayed mainly the same as we described in section 4. However, the biggest change from the original design was that there was not a local vector storing the last read top 5 high scores. Everytime we needed to read or update the high scores, we would read in from the file and then manipulate the data. This caused unnecessary file operations, leading us to implement the vector as a sort of cache, in order to easily insert in a new high score and update the text file appropriately.

User & AI Move Handling

Movement of the user's cursor is working correctly. User is not able to make an invalid move. The blinking of the user/AI moves produces some strange graphical bugs. The terminal cursor moves between two points when flashing, but does not get in the way of what the user is meant to see. Our initial design did not account for the user stack to also need to blink when a move has been chosen. This was solved by reusing the code originally used for the AI move selection blink.

Displaying Pancakes

The pancakes display as they should. Both stacks appear in the same order initially, and contain the same number of pancakes. The pancakes display at an appropriate size, given the available window, and differ in length enough as to be clearly differentiated from one another. The stacks themselves are labeled appropriately as well as being separated by a large enough gap so that both stacks are easily identified. The initial design worked as intended, however some amount of adjustment needed to occur when deciding where on the window the pancakes would be displayed.

AI implementation

When we finished building it, the AI implementation was taking a really long time for difficulty 7 and above. We at first thought that the program was hanging and that it was caused by a segmentation fault however it was just taking way to long. We tried a different implementation using DFS however the time it took to get a favorable move did not change there was just simply too many nodes to look through. We stuck with our design of the AI algorithm described in Section 4.

Section 7 - Conclusions

What Did You Show?

Through this project, we were able to demonstrate a simple technique used to simulate an artificial opponent in our simple video game. We showed how a Minimax tree of an opponent's current game state can be leveraged to select a "good" next move, by comparing all possible future states. We showed how the ncurses API can be leveraged to create an attractive terminal game which relies on computer-human interactions.

What Did You Learn?

There were a lot of things that we learned from this project. We learned about the ncurses API as well as developing a game playing AI. We further improved our usage of github and and further developed our ability to work in a team. We are also now much more familiar with the AGILE and TDD philosophy.

In regards to coordinating the project, we were able to gain experience working on a large software project as a team. We learned how to divide a large problem into logical segments

Burn Down List

Task Number	Task Description	Team Member(s) Responsible	Estimated Hours to Complete	Actual Hours to Complete	Date started	Date completed
1	Design Doc Part 1	Andrew	0.5	0.5	10/14	10/16
2	Design Doc Part 2	Alex	0.5	0.5	10/16	10/17
3	Design Doc Part 3: Usage	David	1	1	10/16	10/17
4	Design Doc Part 3: Models	Stanley	1	1	10/16	10/17
5	Design Doc Part 3: Interactions	Jingchun	1	1	10/16	10/17
6	Design Doc Part 4	Andrew	0.5	0.5	10/14	10/16
7	Design Doc Models	Jingchun	1	3	10/14	10/17
8	Splash Screen	Stanley	1	1	10/21	10/22
9	Test file for Splash screen	Stanley	1	1	11/4	11/4
10	Part 2: Querying Users for Number of Pancakes and Difficulty	David	5	5	10/22	10/22
11	Part 3: User Interface	Alex	1	1	10/21	10/21
12	Part 3: Backend	Alex	1	1	10/21	10/21
13	Part 3: Testing	Alex	2	1	10/21	10/21
14	Part 4: Init highscores file	Andrew	0	1	10/18	10/21
15	Part 4: read in file	Andrew	1	1	10/17	10/21
16	Part 4: Display top 5 scores	Andrew	1	1	10/17	10/21
17	Part 4: Ask User for Initials	Andrew	2	3	10/19	10/21
18	Part 5: Draw Pancakes	David	1	1	10/26	10/26

19	Part 5: Center Pancakes	David	1	1	10/29	10/29
20	Part 6: Test & Dummies	Alex	2	2	11/4	11/4
21	Part 6: Moving Arrow	Alex	1	2	11/4	11/4
22	Part 6: User flip	Alex	1	1	11/4	11/4
23	Part 6: AI blink & flip	Alex	1	1	11/4	11/4
24	Part 7: AI algorithm implementation	Jingchun	4	4	10/25	11/3
25	Part 7: AI testing	Jingchun	3	3	10/25	11/3
26	Part 7: Pancake Data Structure Implementation	Jingchun	0.2	0.2	10/25	10/25
27	Part 7: Pancake Class functionality testing	Jingchun	1	1	10/25	10/25
28	Part 8: write to file	Andrew	1	1	10/21	10/21
29	Part 8: ask user if they wish to play again/loop	Andrew	1	2	10/22	10/23
30	Part 8: add new score to list	Andrew	2	2	10/21	10/23
31	Part 8: Engine/Backend implementation	Jingchun	4	4	11/4	11/4
32	Test High Score File Read	Andrew	1	2	10/31	11/1
33	Test High Score File Write	Andrew	1	2	10/31	11/1
34	Test High Score Add	Andrew	2	2	11/2	11/3
35	Extra credit 1: Splash Animation	Stanley	1	1	11/1	11/5
36	Extra Credit: Hint Button	Andrew	1	3	11/4	11/5
37	Extra Credit: Minimum Moves	Andrew	1	3	11/4	11/5
38	Compile/Link Using Grader's Flags	Andrew	1	2	11/4	11/4
39	Fix Memory Leaks	Everyone	1	4	11/4	11/5

Section 8 - Future Research

The biggest challenge facing our program is the overall speed of the AI. The AI functions incredibly well, but the speed with which it searches through the AI tree is rather slow. Currently the search function takes big-O time to complete, or time. This is far too slow for a program to be used by the public.

For future research we should look into algorithms that will speed up searching through the AI tree. We believe that informed searching should be implemented and a large majority of our future research should be dedicated to this topic. Informed searching has the ability to greatly increase the speed of our AI searching speed. It does require a little more overhead, but we feel that this overhead will not slow down the new algorithm enough as to render it mute.

Another area we should improve upon is the overall aesthetic of the program. The use of ncurses is a relatively new topic for us and us such was not utilized as well as it could have been. Our program looks rather elementary as far as the general design, and more research into ncurses might provide our group with a better understanding of how to create a more aesthetically pleasing user display.

Section 9 - Instructions

We provide some detailed instructions on how to get started with building and playing our game. Although the process is simple, we hope to prevent the reader from encountering any confusing situations when using our project.

Building

We assume the reader has access to our source code. To build the project, we utilize the *make* build automation tool [1], so this step is easy. Simply run the command `make all`. This should create 2 executable files: `game` and `test`.

Running

After following the steps above, you can either play the game, or run our automated test script. To play the game, run the command `./game`. To run the test script, run the command `./test`.

Section 10 - Listing of the Program

Main.cpp

```
#include "Screens.h"


int main() {
  Screens a = Screens();
  a.setup();
  while (a.keep_playing()) {
    a.instructions();
    a.create_menu();
    a.set_pancake_order();
    a.enter_initials();
    a.play_game();
    a.post_game();
  }
  a.teardown();
}
```

Screens.h

```
#ifndef
SCREENS_H
        #define SCREENS_H


        #include <fstream>
        #include "Engine.h"
```

```
using namespace std;


class Screens {
  public:
      // create Game object
      Screens();
      // init ncurses and show splashscreen (called at init)
      void setup();
      // destroy the game window and return to std in/out (called at termination)
      void teardown();



//================================================
=========================================
      // Parts 1 & extra credit 1 (splash screen + demo gameplay & instructions)
      // display the splash screen and the blinking prompt
      void splashscreen();
      //display the initial state of the game demo
      void print_state1();
      //display the blinking prompt of the game demo
      void print_state2();
      //display the final state of the game demo
      void print_empty_state();
      //display the animation of the game demo on the splash screen
      void animation();
      // prompt the user to press enter to start the game
      void blink_prompt();
      // screen displaying instructions for how to play the game
```

```
        void instructions();
        // verify splash screen displays correctly
        bool test_splashscreen();



//================================================
=========================================
        // Parts 2 (setting pancake stack size and game difficulty)
        // prompt user to configure size and difficulty
        void create_menu();



//================================================
=========================================
        //Part 3 (random or assigning stack order)
        void set_pancake_order();
        //sets order of pancake stack in engine object



        //Stack order selection tests.
        bool test_random_stack_order();
        bool test_user_stack_duplicate_check();
        bool test_user_stack_range_check();



//================================================
=========================================
        // Parts 4 & 8 (FILE I/O & TOP SCORES)
        // prompt user to enter their initials and then display top 5 scores
        void enter_initials();
        // screen for actual gameplay
        void play_game();
```

```
    // calculate game score, display results, and update highscores.txt
    void post_game();
    // prompt user to play another game or end the program
    bool keep_playing();
    // verify reading top scores from a file
    bool test_file_read();
    // verify writing high scores to a file
    bool test_file_write();
    // verify adding 2 scores to a list of top scores
    // one score should be added to position 3,
    // and the other score should not be added
    bool test_add_score();



//================================================
=========================================
    //Part 6 tests
    bool test_arrow_move();

    // TEST FUNCTIONS
    void test_player_selection();
    void test_player_move();
    void test_blink_stack();
    bool determine_box_order_test();
    bool center_output_test();


  private:
    Engine game; // the game engine controlling gameplay
    bool done; // signals if the user is ready to terminate the game
```

```
//================================================
==================================
    // Part 2 Helper functions
    // sanitize input for selecting pancake stack size
    int check_pancakes();
    // sanitize input for selecting game difficulty
    int check_diff(int num_pancakes);
    // print the order of the vector to the terminal std out
    void print_user_order(vector<int> user_order, int num_pancakes);
    // print selected difficulty to the terminal std out
    void print_user_diff(int diff);



//================================================
==================================
    //Part 3 Helper functions
    vector<int> get_user_order(int num_pancakes);
    //asks user for random or assigned pancake stack order
    vector<int> random_stack_order(int num_pancakes);
    //returns a random order of pancakes given number of pancakes
    vector<int> user_stack_order(int num_pancakes);
    //takes user input for order of pancake stack
    bool user_stack_duplicate_check(vector<int> order, int user_choice, int y, int
x);
    //checks that user's input is not a duplicate pancake
    bool user_stack_range_check(int user_choice, int num_pancakes, int y, int
x);
    //checks that user's input is within valid pancake sizes
    int user_stack_print_prompt(int num_pancakes, int y, int x);
    //prints prompt for user assignment
```

```
//================================================
=======================================
    //  Parts 4 & 8 Helper functions
    // display the top 5 scores to the screen
    void display_top_5();
    // read top 5 scores from file and store them locally in a vector
    void read_top_5();
    // write high_scores into the file highscores.txt
    void write_top_5();
    // return index where score should be added into vector high_scores
    // return -1 if the score should not be added
    int in_top_5(int score);
    // add score to vector high_scores if in_top_5 returns true
    void add_score(tuple<string, int, int> score);
    // create the highscore.txt and fill with '--- 0 0'
    void init_high_score_file();
    // on game completion ask the user if they would like to play again
    // consequently changes the variable done
    void play_again();
    // save the contents of highscores.txt to a vector and return it
    vector<tuple<string, int, int>> save_file_contents();
    // restore highscores.txt with scores from a vector
    void restore_file_contents(vector<tuple<string, int, int>> v);
    // create and return a test vector with sample data, used for testing
    vector<tuple<string, int, int>> make_test_vector();



//================================================
==================================
```

```
        //Part 6 & 7 (controls for gameplay & calling minimax tree functions in class
aiTree)
        int arrow_move(int y, int x, int cursor_pos, int num_pancakes, int move);
        //handles individual moves of user cursor
        int player_selection(int y, int x);
        //handles user's moving of arrow and selection of a pancake to flip
        void player_move();
        //handles user's move input, flipping of stack in engine, and reprinting of
user's stack
        void ai_move();
        //handles ai output for move selection, flipping of ai stack in engine, and
reprinting of ai stack
        void blink_stack(int y, int x, vector<int> sub_stack);
        //blinks the portion of the pancake stack for a given move
        void arrow_init(int y, int x, int cursor_pos);
        //prints first arrow on screen
        void erase_pancakes(int top_side, int left_side, int num_cols, int
num_pancakes);
        //clears a portion of the screen for a new pancake stack printing


//=================================================
================================
        //Part 5 (draw centered pancakes)
        //Wrappers to create vectors that have strings representing pancakes
        vector<string> box_1_wrapper();
        vector<string> box_2_wrapper();
        vector<string> box_3_wrapper();
        vector<string> box_4_wrapper();
        vector<string> box_5_wrapper();
        vector<string> box_6_wrapper();
        vector<string> box_7_wrapper();
        vector<string> box_8_wrapper();
```

```
        vector<string> box_9_wrapper();
        // pad a string str with spaces to center it within num_cols
        void center_output(string str, int num_cols);
        // draw pancakes given a stack and a top-left coordinate point
        void draw_pancakes(vector<int> stack, int top_side, int left_side);

        vector<vector<string> > determine_box_order(vector<int> stack);
        // print 'PLAYER' on top of player's pancake stack
        void label_user_stack(int top_side, int left_side);
        // print 'AI' on top of AI's pancake stack
        void label_ai_stack(int top_side, int left_side);
        //Part 5 Test Helpers
        vector<vector<string> > box_order_test_comparison_vector();
        vector<int> box_order_test_vector();

};

#endif
```

Screens.cpp

```
#include
"Screens.
h"
#include <algorithm>
#include <ctime>
#include <stdlib.h>
#include <iostream>
#include <csignal>
```

```
#include <unistd.h>
#include <string.h>
using namespace std;


Screens::Screens() {
  game = Engine();
  done = false;
}


void Screens::setup() {
  read_top_5();
  initscr();
  noecho();
  cbreak();
  keypad(stdscr, TRUE);
  splashscreen();
  blink_prompt();
  instructions();
}


void Screens::teardown() {
  endwin();
}


// part
1==================================================
======================
void Screens::splashscreen() {
  initscr();
```

```
    noecho();
    start_color();
    init_pair(1, COLOR_RED, COLOR_BLACK);
    attron(COLOR_PAIR(1));
    mvaddstr(16, 20, "                    |           ");
    mvaddstr(17, 20, " ___  ___  ___  ___  ___ |    ___  ___");
    mvaddstr(18, 20, "|   )|   )|   )|    |   )|___)|___)|___ ");
    mvaddstr(19, 20, "|__/ |__/||  / |__  |__/|| \\  ||__   __/ ");
    mvaddstr(20, 20, "|");
    attroff(COLOR_PAIR(1));
    mvaddstr(23, 33, "Team name: 13");
    mvaddstr(24, 33, "Team members: ");
    mvaddstr(25, 33, "JingChun Qu");
    mvaddstr(26, 33, "Andrew Chin");
    mvaddstr(27, 33, "Alexander Staggs");
    mvaddstr(28, 33, "Zicheng Chen");
    mvaddstr(29, 33, "David Tieding");
}


void Screens::print_state1() {
    mvprintw(6, 34, " +-----+  ");
    mvprintw(7, 34, " |  2  |  ");
    mvprintw(8, 34, " +-----+  ");
    mvprintw(9, 34, "  +---+   ");
    mvprintw(10, 34, "  | 1 |   ");
    mvprintw(11, 34, "  +---+   ");
    mvprintw(12, 34, "+-------+");
    mvprintw(13, 34, "|   3   |");
    mvprintw(14, 34, "+-------+");
}
```

```
void Screens::print_state2() {
  mvprintw(6, 34, "  +---+   ");
  mvprintw(7, 34, "  | 1 |   ");
  mvprintw(8, 34, "  +---+   ");
  mvprintw(9, 34, " +-----+  ");
  mvprintw(10, 34, " |  2  |  ");
  mvprintw(11, 34, " +-----+  ");
  mvprintw(12, 34, "+-------+");
  mvprintw(13, 34, "|   3   |");
  mvprintw(14, 34, "+-------+");
}

void Screens::print_empty_state() {
  for (int i = 6; i < 12; ++i)
    mvprintw(i, 34, "                       ");
  mvprintw(12, 34, "+-------+");
  mvprintw(13, 34, "|   3   |");
  mvprintw(14, 34, "+-------+");
}

void Screens::animation() {
  print_state1();
  refresh();
  napms(1000);
  for (int i = 0; i < 3; ++i) {
    print_state1();
    refresh();
    napms(500);
```

```
        print_empty_state();
        refresh();
        napms(500);

    }
    print_state2();
    refresh();
    napms(5000);
}



void Screens::blink_prompt() {
    nodelay(stdscr, TRUE);
    attron(A_BLINK);
    mvwprintw(stdscr, 35, 26, "Press any key to continue!!!");
    attroff(A_BLINK);
    refresh();


    while (1) {
        animation();
        int ch = getch();
        if (ch == ERR || ch != 10) {
            continue;
        }
        else {
            break;
        }
        for (int c = 6; c<12 ; c++) {
            move(c, 20);
```

```
            clrtoeol();
            refresh();

        }
        refresh();
    }
    nodelay(stdscr, FALSE);
}


void Screens::instructions() {
    erase();

printw("============================================================
==================================\n");
    printw("                                HOW TO PLAY\n");

printw("============================================================
==================================\n");
    printw("   Pancakes is simple and fun to play. Your goal is to stack your pancakes
in\n");
    printw(" increasing order, with the smallest pancake on top. Additionally, there is
an\n");
    printw(" added challenge: you need to stack your pancakes before the AI stacks its
own!\n\n");
    printw(" The rules are simple: you may only flip over n pancakes from the top in
order\n");
    printw("                             to sort your stack.\n\n\n");
    printw("                     CONTROLS (during stacking gameplay):\n");
    printw("   ------------------------------------------------------------------------\n");
    printw("    -     W : move your cursor up\n");
    printw("    -     S : move your cursor down\n");
    printw("    - ENTER : flip the top n pancakes, starting from your cursor and
upwards\n");
    printw("    -     H : display a hint\n\n");
```

```
  printw("                          _________________\n");
  printw("                         |                 |\n");
  printw("                         | Good luck!!! |\n");
  printw("                         |_________________|\n\n\n\n");
  printw("                      Press any key to continue.");
  getch();
}


bool Screens::test_splashscreen() {
  splashscreen();
  char test1 =mvinch(10, 29);
  char test2 = mvinch(14, 30);
  endwin();
  if(test1 == 'T' && test2 == 'Z') {
     return true;
  }
  else {
     return false;
  }
}
// end of part
1==================================================
=================


// part
2==================================================
=========================
void Screens::create_menu() {
  erase();
  int difficulty = 0, num_pancakes = 0;
  move(0, 0);
```

```
    clrtoeol();
    printw("Specify the Number of Pancakes You Want: ");
    num_pancakes = check_pancakes();
    game.set_num_pancakes(num_pancakes);
    printw("\n\nStacks will contain %d pancakes.\n\nPress any key to continue.",
num_pancakes);
    getch();
    erase();
    move(0, 0);
    clrtoeol();
    printw("Specify the Difficulty of the Game [1-%d]: ", num_pancakes);
    difficulty = check_diff(num_pancakes);
    game.set_difficulty(difficulty);
    printw("\n\nGame will be played on difficulty %d.\n\nPress any key to continue.",
difficulty);
    getch();
    erase();
}


int Screens::check_pancakes() {
    char c = getch();
    string n = string(1, c);
    while (n.at(0) < 50 || n.at(0) > 57) {
        printw("\nNumber of pancakes must be an integer greater than 1.\n");
        printw("Specify the Number of Pancakes You Want: ");
        c = getch();
        n = string(1, c);
    }
    return stoi(n);
}
```

```
int Screens::check_diff(int num_pancakes) {
  char c = getch();
  string diff = string(1, c);
  int n_ascii = num_pancakes + 48;
  while (diff.at(0) < 49 || diff.at(0) > n_ascii) {
     printw("\nDifficulty must be in the interval.\n");
     printw("Specify the Difficulty of the Game [1-%d]: ", num_pancakes);
     c = getch();
     diff = string(1, c);
  }
  return stoi(diff);
}


void Screens::print_user_order(vector<int> user_order, int num_pancakes) {
  cout << "Vector order: " << endl;
  for (int i = 0; i < num_pancakes; i++) {
     cout << user_order.at(i) << " ";
  }
  cout << endl;
}


void Screens::print_user_diff(int diff) {
  cout << "User difficulty is: " << diff << endl;
}
// end of part
2================================================================
===============
```

```
/*============================================================
============================
* Alex's Part, Parts 3 & 6
To test, at each screen, try invalid inputs and see if proper error messages appear.
If it runs successfully with invalid inputs, then try a run without any invalid inputs.
Invalid inputs should include numbers not expected and random keystrokes (like space or arrows).
*/


void Screens::set_pancake_order() {
  game.set_order(get_user_order(game.get_num_pancakes()));
}


//can't simulate user inputs
vector<int> Screens::get_user_order(int num_pancakes) {
  erase();
  mvprintw(0, 0, "Random or Assign stack order? (Press 1 for random, 2 for assign)");
  refresh();
  int rand_selection;
  while (1) {  //returns break this loop rand_selection = getch();//1 for random, 2 for assign
      rand_selection = getch();  //1 for random, 2 for assign
      if (rand_selection == 49) {  //1 in ascii is 49
          mvprintw(1, 0, "Random order selected.\n");
          printw("Press any key to continue.");
          getch();
          return random_stack_order(num_pancakes);
      }
      else if (rand_selection == 50) {
          return user_stack_order(num_pancakes);
```

```
        }
        else {
            move(1, 0);
            mvprintw(1, 0, "Press 1 or 2 for your selection.");
        }
    }
}


//can't simulate inputs, no test
vector<int> Screens::user_stack_order(int num_pancakes) {  //get order helper for user choice
    vector<int> order;
    int x = 0, y = 1;
    y = user_stack_print_prompt(num_pancakes, y, x);  //prints prompt and returns next line
    int user_choice;
    for (int i = 0; i < num_pancakes; i++) {
        move(y, x);
        printw("Input size of Pancake %d. (Press a number 1-%d)", i, num_pancakes);
        user_choice = getch() - 48;  //converts from char to int
        if (user_stack_range_check(user_choice, num_pancakes, y, x) &&
                user_stack_duplicate_check(order, user_choice, y, x)) {  //check inputs
            move(++y, x);
            clrtoeol();
            printw("Size of pancake %d is: %d", i, user_choice);
            order.push_back(user_choice);
            y++;
        }
        else
            i--;  //retry i iter
    }
```

```
  y++;
  move(y, x);
  return order;
}


int Screens::user_stack_print_prompt(int num_pancakes, int y, int x) {  //returns
next line to move to
  //test would be trivial.
  move(y, x);
  clrtoeol();
  printw("Press numbers 1-%d to assign pancake sizes:", num_pancakes);
  y++;
  move(y, x);
  printw("[Top pancake is Pancake 0.]");
  y++;
  return y;
}


bool Screens::test_user_stack_range_check() {
  bool all_passed = true;
  initscr();
  for (int i = 1; i <= 9; i++) {
     if (!user_stack_range_check(i, 9, 0, 0))
        all_passed = false;
  }
  bool bad_check = user_stack_range_check(5, 4, 0, 0);  //should be false;
  string expected_string = "Press a number 1-4.";
  char actual[100];
  mvinnstr(1, 0, actual, expected_string.length());
  string actual_string = actual;
```

```
  if (expected_string.compare(actual_string) != 0 || bad_check)
      all_passed = false;
  endwin();
  return all_passed;
}


bool Screens::user_stack_range_check(int user_choice, int num_pancakes, int y, int
x) {
  //checks to see if user input for pancake size is within valid range
  if (!(user_choice > 0 && user_choice <= num_pancakes)) {
      move(y+1, x);
      clrtoeol();
      printw("Press a number 1-%d.", num_pancakes);
      return false;
  }
  return true;
}


bool Screens::test_user_stack_duplicate_check() {
  initscr();
  bool all_passed = true;
  vector<int> order;
  order.push_back(2);
  order.push_back(1);
  bool dup_check = user_stack_duplicate_check(order, 2, 0, 0);  //should be false
  string expected_string = "A pancake of size 2 is already in the stack. Try again.";
  char actual[100];
  mvinnstr(1, 0, actual, expected_string.length());
  string actual_string = actual;
  if (expected_string.compare(actual_string) != 0 || dup_check)
```

```
        all_passed = false;
    dup_check = user_stack_duplicate_check(order, 3, 0, 0);  //should be true
    if (!dup_check)
        all_passed = false;
    endwin();
    return all_passed;  //result
}


bool Screens::user_stack_duplicate_check(vector<int> order, int user_choice, int y,
int x) {
  //checks to see if user tries to make two pancakes the same size
  //true if no duplicates, false if duplicates
  bool duplicate = false;
  for (int j = 0; j < (int)order.size(); j++) {  //check for duplicate input
      if (order.at(j) == user_choice) {
          duplicate = true;
      }
  }
  if (duplicate) {  //print error message
      move(y+1, x);
      clrtoeol();
      printw("A pancake of size %d is already in the stack. Try again.", user_choice);
  }
  return !duplicate;
}


bool Screens::test_random_stack_order() {
  bool all_passed = true;
  for (int i = 2; i <=9; i++) {  //each stack size
      bool found;
```

```
        bool valid = true;
        vector<int> random_stack = random_stack_order(i);
        for (int j = 0; j < (int)random_stack.size(); j++) {  //each pancake size
            found = false;
            for (int k = 0; k < (int)random_stack.size(); k++) {  //each pancake
                if (j+1 == random_stack.at(k))
                    found = true;
            }
            if (found == false)
                valid = false;
        }
        cout << "Random Stack test " << i;
        if (valid)
            cout << " passed." << endl;
        else {
            cout << " failed." << endl; all_passed = false;
        }
    }
    return all_passed;
}


vector<int> Screens::random_stack_order(int num_pancakes) {  //get order helper
for rand order
    vector<int> order;
    for (int i = 0; i < num_pancakes; i++) {  //random order
        order.push_back(i+1);
    }
    srand(time(0));  //time seed to make truly random
    random_shuffle(order.begin(), order.end());
    move(2, 0);
```

```
    return order;
}
// end of part
3==================================================
==================


//================================================
================================
// Part 6, user move, AI blink, flip printing
bool Screens::test_arrow_move() {
    bool all_passed = true;
    initscr();
    string expected = "->";
    int move_output = arrow_move(0, 0, 1, 3, 1);  //test going above limit
    char actual_cstr[2];
    mvinnstr(1, 0, actual_cstr, expected.length());
    string actual = actual_cstr;
    if (expected.compare(actual) != 0 || move_output != 1) {
        all_passed = false;
        cout << "test 1 failed";
    }
    move_output = arrow_move(0, 0, 3, 3, -1);  //test going above limit
    mvinnstr(7, 0, actual_cstr, expected.length());
    actual = actual_cstr;
    if (expected.compare(actual) != 0 || move_output != 3) {
        all_passed = false;
        cout << "test 2 failed";
    }
    move_output = arrow_move(0, 0, 3, 3, 0);
    if (move_output != 0) {
        all_passed = false;
```

```
        cout << "test 3 failed";
    }
    endwin();
    return all_passed;
}


//no way to simulate key presses, so second optional arguments for testing
int Screens::arrow_move(int y, int x, int cursor_pos, int num_pancakes, int move) {
    //move int determines move, -1 = down, 1 = up, 0 = enter
    //return pos if enter, 0 if move cursor
    for(int i = y; i < y+num_pancakes*3; i++) {  //clear old arrow
        mvprintw(i, x, "  ");
    }
    if(move == 0) {  //skip if enter key pressed
        return 0;
    }
    else if(move == 1 || move == -1) {
        if((move == 1 && cursor_pos == 1) || (move == -1 && cursor_pos ==
num_pancakes-1))
            move = 0;  //handle arrow boundaries
        int new_y = y+1+3*(cursor_pos-1-move);
        mvprintw(new_y, x, "->");
        return cursor_pos-move;
    }
    else {
        cout << "Unexpected Error Occurred.";
        return -1;
    }
}
```

```
void Screens::arrow_init(int y, int x, int cursor_pos) {
  mvprintw(y+1+3*(cursor_pos-1), x, "->");
}


void Screens::test_player_selection() {
  //can't simulate user input, manual test
  initscr();
  vector<int> order;
  order.push_back(4);
  order.push_back(2);
  order.push_back(1);
  order.push_back(3);
  game.set_order(order);
     int sel = player_selection(0, 0);
     endwin();
     cout << "player selection was: " << sel << endl;
}


int Screens::player_selection(int y, int x) {
  noecho();
  keypad(stdscr, TRUE);
  int num_pancakes = game.get_num_pancakes();
  int cursor_pos = 1;  //cursor position
  arrow_init(y, x, cursor_pos);
  int key_input, output_pos, move;
  while (true) {  //cursor_pos == 0 means select location
     key_input = getch();
     if (key_input == 104)  // hint
        mvprintw(1, 0, "Try flipping the top %d pancakes.", game.get_hint());
```

```
    else if (key_input == 10 || key_input == 'w' || key_input == 's') {  //enter, w,
s
        if (key_input == 'w') move = 1;
        else if (key_input == 's') move = -1;
        else if (key_input == 10) move = 0;  //can be optimized to return at move =
0
        output_pos = arrow_move(y, x, cursor_pos, num_pancakes, move);
        if (output_pos == 0) {
            return cursor_pos;
        }
        else
            cursor_pos = output_pos;
    }
  }
}


void Screens::player_move() {
  int y = 5;
  int x = 10;
  int flip_location = player_selection(y+3, x);
  vector<int> user_order = game.get_user_order();
  vector<int> sub_stack;
  for (int i = 0; i <= flip_location; i++) {
     sub_stack.push_back(user_order.at(i));
  }
  blink_stack(y, x, sub_stack);
  refresh();
  game.flip_user(flip_location);
  erase_pancakes(y, x, 30, game.get_num_pancakes());
  draw_pancakes(game.get_user_order(), y, x);
  refresh();
```

```
}


void Screens::erase_pancakes(int top_side, int left_side, int num_cols, int
num_pancakes) {
  for (int i = top_side; i < 3*num_pancakes+top_side; i++) {
     for (int j = left_side; j < num_cols+left_side; j++) {
        mvprintw(i, j, " ");
     }
  }
}


void Screens::test_blink_stack() {
  initscr();
  vector<int> test_vec;
  test_vec.push_back(2);
  test_vec.push_back(1);
  test_vec.push_back(3);
  blink_stack(0, 0, test_vec);
  endwin();
}


void Screens::blink_stack(int y, int x, vector<int> sub_stack) {
  //mvprintw(30, 0, "Blink Called");
  refresh();
  for (int i = 0; i < 3; i++) {
     erase_pancakes(y, x, 30, sub_stack.size());
     refresh();
```

```
        napms(500);
        draw_pancakes(sub_stack, y, x);
        refresh();
        napms(500);
    }
}


void Screens::ai_move() {
    //mvprintw(29, 0, "ai move called");
    refresh();
    int y = 5;
    int x = 40;
    int ai_move = game.get_ai_move();
    //mvprintw(31, 0, "get_ai_move terminated");
    refresh();
    vector<int> ai_order = game.get_ai_order();
    vector<int> sub_stack;
    for (int i = 0; i <= ai_move; i++) {
        sub_stack.push_back(ai_order.at(i));
    }
    blink_stack(y, x, sub_stack);
    refresh();
    game.flip_ai(ai_move);
    erase_pancakes(y, x, 30, game.get_num_pancakes());
    draw_pancakes(game.get_ai_order(), y, x);
}
//end of part 6
//================================================
=========================================
```

```
// part 4 and
8==========================================================
==================================
void Screens::enter_initials() {
  erase();
  string initials;
  char str[80];
  echo();
  while (true) {
     printw("Please enter your initials: ");
     getstr(str);
     initials = string(str);
     if (initials.length() == 3)
        break;
     printw("Invalid initials (must be 3 characters).\n");
  }
  noecho();
  game.set_initials(initials);
  display_top_5();
  printw("\n%s (Lv.%d) %d\n", initials.c_str(), game.get_difficulty(), 0);
  printw("\nPress any key to continue.");
  getch();
  refresh();
}


void Screens::post_game() {
  erase();
  move(0, 0);
  int curr_score = game.calculate_score(game.get_num_pancakes());
  int curr_difficulty = game.get_difficulty();
  string initials = game.get_initials();
```

```
    int pos = in_top_5(curr_score);
    printw("%s\n[%s] On difficulty %d, you scored %d pts.\n\n",
game.get_winner().c_str(), initials.c_str(), curr_difficulty, curr_score);
    if (pos == -1) {
        printw("Sorry, your score is not one of the top 5 highest scores.\n\n");
        display_top_5();
        printw("\n%s (%d) %d\n", initials.c_str(), curr_difficulty, curr_score);
    }
    else {
        printw("You made it onto the leaderboard!\n\n");
        add_score(make_tuple(initials, curr_difficulty, curr_score));
        display_top_5();
        write_top_5();
    }
    play_again();
}


void Screens::display_top_5() {
    vector<tuple<string, int, int>> high_scores = game.get_top_5();
    printw("Top 5 High Scores\n");
    printw("-----------------\n");
    for (auto t : high_scores) {
        printw("%s (%d) %d\n", get<0>(t).c_str(), get<1>(t), get<2>(t));
    }
}


void Screens::read_top_5() {
    vector<tuple<string, int, int>> top5;
    string initials;
    int difficulty, score;
```

```
    ifstream file("highscores.txt");
    if (!file) {
        init_high_score_file();
        file.open("highscores.txt");
    }
    while (file >> initials >> difficulty >> score) {
        top5.push_back(make_tuple(initials, difficulty, score));
    }
    file.close();
    game.set_high_scores(top5);
}


void Screens::write_top_5() {
    vector<tuple<string, int, int>> high_scores = game.get_top_5();
    ofstream file("highscores.txt");
    for (int i = 0; i < 5; ++i) {
        file << get<0>(high_scores.at(i)) << " " << get<1>(high_scores.at(i)) << " "
<< get<2>(high_scores.at(i)) << endl;
    }
    file.close();
}


int Screens::in_top_5(int score) {
    vector<tuple<string, int, int>> high_scores = game.get_top_5();
    if (!high_scores.empty()) {
        for (int i = 0; i < 5; ++i) {
            if (score > get<2>(high_scores.at(i)))
                return i;
        }
        return -1;
```

```
    }
    return 0;
}
void Screens::add_score(tuple<string, int, int> score) {
    vector<tuple<string, int, int>> high_scores = game.get_top_5();
    if (!high_scores.empty()) {
        int pos = in_top_5(get<2>(score));
        if (pos != -1) {
            for (int i = 4; i > pos; --i) {
                high_scores.at(i) = high_scores.at(i - 1);
            }
            high_scores.at(pos) = score;
        }
    }
    game.set_high_scores(high_scores);
}


void Screens::init_high_score_file() {
    ofstream file("highscores.txt");
    for (int i = 0; i < 5; ++i) {
        file << "--- 0 0" << endl;
    }
    file.close();
}


void Screens::play_again() {
    cbreak();
    printw("\nKeep playing? (y/n): ");
    char ans = getch();
```

```
  while (true) {
    if (ans == 110) {
      done = true;
      return;
    }
    else if (ans == 121) {
      erase();
      return;
    }
    else {
      move(14, 0);
      printw("Enter y or n: ");
      ans = getch();
    }
  }
}


bool Screens::keep_playing() {
  return !done;
}


bool Screens::test_file_read() {
  vector<tuple<string, int, int>> save = save_file_contents();
  ofstream file("highscores.txt");
  file << "AAA 4 10000" << endl << "BBB 5 9000" << endl << "CCC 8 5500" <<
endl << "DDD 7 4000" << endl << "EEE 3 100" << endl;
  file.close();
  read_top_5();
  vector<tuple<string, int, int>> expected = make_test_vector();
  for (int i = 0; i < 5; ++i) {
```

```
        vector<tuple<string, int, int>> v = game.get_top_5();
        if (get<0>(v.at(i)) != get<0>(expected.at(i)) || get<1>(v.at(i)) !=
get<1>(expected.at(i)) || get<2>(v.at(i)) != get<2>(expected.at(i))) {
            restore_file_contents(save);
            return false;
        }
    }
    restore_file_contents(save);
    return true;
}


bool Screens::test_file_write() {
    vector<tuple<string, int, int>> save = save_file_contents();
    vector<tuple<string, int, int>> test = make_test_vector();
    game.set_high_scores(test);
    write_top_5();
    vector<string> expected = {"AAA 4 10000", "BBB 5 9000", "CCC 8 5500", "DDD 7
4000", "EEE 3 100"};
    ifstream file("highscores.txt");
    string line;
    for (int i = 0; i < 5; ++i) {
        getline(file, line);
        if (line != expected.at(i)) {
            restore_file_contents(save);
            return false;
        }
    }
    restore_file_contents(save);
    return true;
}
```

```
bool Screens::test_add_score() {
  vector<tuple<string, int, int>> save = save_file_contents();
  vector<tuple<string, int, int>> test = make_test_vector();
  game.set_high_scores(test);
  add_score(make_tuple("NEW", 5, 6500));
  add_score(make_tuple("BAD", 8, 0));
  test = game.get_top_5();
  vector<tuple<string, int, int>> expected;
  expected.push_back(make_tuple("AAA", 4, 10000));
  expected.push_back(make_tuple("BBB", 5, 9000));
  expected.push_back(make_tuple("NEW", 5, 6500));
  expected.push_back(make_tuple("CCC", 8, 5500));
  expected.push_back(make_tuple("DDD", 7, 4000));
  for (int i = 0; i < 5; ++i) {
     if (expected.at(i) != test.at(i)) {
        restore_file_contents(save);
        return false;
     }
  }
  restore_file_contents(save);
  return true;
}


vector<tuple<string, int, int>> Screens::save_file_contents() {
  read_top_5();
  return game.get_top_5();
}
```

```
void Screens::restore_file_contents(vector<tuple<string, int, int>> v) {
  game.set_high_scores(v);
  write_top_5();
}


vector<tuple<string, int, int>> Screens::make_test_vector() {
  vector<tuple<string, int, int>> test;
  test.push_back(make_tuple("AAA", 4, 10000));
  test.push_back(make_tuple("BBB", 5, 9000));
  test.push_back(make_tuple("CCC", 8, 5500));
  test.push_back(make_tuple("DDD", 7, 4000));
  test.push_back(make_tuple("EEE", 3, 100));
  return test;
}
// end of parts 4 and
8================================================
=============


// part
7================================================
============================
void Screens::play_game() {
  echo();
  erase();
  label_user_stack(3, 10);
  label_ai_stack(3, 40);
  draw_pancakes(game.get_user_order(), 5, 10);
  draw_pancakes(game.get_user_order(), 5, 40);
  noecho();
  while (!game.game_over()) {
```

```
        mvprintw(0, 0, "Minimum Number of Steps to Solve: %d",
game.get_min_moves());
        mvprintw(1, 0, "[Need a hint? Press 'h']            ");
        player_move();
        mvprintw(1, 0, "Opponent is thinking...             ");
        ai_move();
        refresh();
    }
    printw("\n\n                Game finished. Press any key to continue.");
    getch();
}
// end of part
7=================================================
=====


// part
5=================================================
============================
void Screens::center_output(string str, int num_cols) { //Allows the pancakes to be
displayed appropriately
    int padding_left = (num_cols/ 2) - (str.size() / 2); //Makes sure the pancakes are
centered relative to one another

    for(int i = 0; i < padding_left; i++) {
        printw(" ");
    }


    printw(str.c_str());
}


bool Screens::center_output_test() { //Tests to ensure that the pancakes are
actually being centered relative to one another
```

```
  int top_side = 0;
  vector<string> test_string = box_1_wrapper();

  move(top_side, 0);

  for(int i = 0; i < 3; i++) {
     center_output(test_string[i], 30);
     move(top_side += 1, 0);
  }

  char test1 = mvinch(0, 0);
  char test2 = mvinch(0, 2);
  char test3 = mvinch(0, 4);
  char test4 = mvinch(1, 0);
  char test5 = mvinch(1, 2);
  char test6 = mvinch(1, 4);
  endwin();
  if(test1 == '+' && test2 == '-' && test3 == '+' && test4 == '|' && test5 == '1' &&
test6 == '|') {
     return true;
  }
  else {
     return false;
  }
}


//Wrappers that contain vectors of strings comprised of the individual elements of
the pancakes
vector<string> Screens::box_9_wrapper() {
  vector<string> box_9 = {
  "+------------------+",
```

```
  "|       9       |",
  "+-----------------+"
  };
  return box_9;
}


vector<string> Screens::box_8_wrapper() {
  vector<string> box_8 = {
  "+-----------------+",
  "|       8       |",
  "+-----------------+"
  };
  return box_8;
}




vector<string> Screens::box_7_wrapper() {
  vector<string> box_7 = {
  "+---------------+",
  "|      7      |",
  "+---------------+"
  };
  return box_7;
}


vector<string> Screens::box_6_wrapper() {
  vector<string> box_6 = {
  "+-------------+",
```

```
  "|     6     |",
  "+-------------+"
  };
  return box_6;
}


vector<string> Screens::box_5_wrapper() {
  vector<string> box_5 = {
  "+-----------+",
  "|    5    |",
  "+-----------+"
  };
  return box_5;
}


vector<string> Screens::box_4_wrapper() {
  vector<string> box_4 = {
  "+---------+",
  "|   4   |",
  "+---------+"
  };
  return box_4;
}


vector<string> Screens::box_3_wrapper() {
  vector<string> box_3 = {
  "+-------+",
  "|  3  |",
  "+-------+"
```

```
  };
  return box_3;
}


vector<string> Screens::box_2_wrapper() {
  vector<string> box_2 = {
  "+-----+",
  "|  2  |",
  "+-----+"
  };
  return box_2;
}


vector<string> Screens::box_1_wrapper() {
  vector<string> box_1 = {
  "+---+",
  "| 1 |",
  "+---+"
  };
  return box_1;
}

void Screens::label_user_stack(int top_side, int left_side) { //Indicates which stack is the users
  move(top_side, left_side);
  center_output("PLAYER", 30);
}

void Screens::label_ai_stack(int top_side, int left_side) { //Indicates which stack is the AIs
```

```
  move(top_side, left_side);
  center_output("AI", 30);
}

void Screens::draw_pancakes(vector<int> stack, int top_side, int left_side) {
//draws the appropriate pancakes in the appropriate places
  int num_cols = 30;
  move(top_side, left_side); //Allows us to print the pancakes anywhere on the
screen that we wish
  vector<vector<string>> boxes = determine_box_order(stack); //Determines
which pancakes and which order those pancakes should be displayed in

  for (int i = 0; i < (int)boxes.size(); i++) { //Prints the pancakes appropriately
    for (int j = 0; j < 3; j++) {
      center_output(boxes[i][j], num_cols);
      move(top_side+=1, left_side);
    }
  }
}

vector<vector<string>> Screens::determine_box_order(vector<int> stack) {
//Determines the correct order for the pancakes to be displayed in
  //Brad said leaving it as 34 lines was ok because breaking up a switch statement is
gross
  vector<vector<string>> boxes;
  for(int i = 0; i < (int)stack.size(); i++) {
    switch(stack[i]) {
      case 1:
        boxes.push_back(box_1_wrapper());
        break;
      case 2:
        boxes.push_back(box_2_wrapper());
        break;
```

```
            case 3:
                boxes.push_back(box_3_wrapper());
                break;
            case 4:
                boxes.push_back(box_4_wrapper());
                break;
            case 5:
                boxes.push_back(box_5_wrapper());
                break;
            case 6:
                boxes.push_back(box_6_wrapper());
                break;
            case 7:
                boxes.push_back(box_7_wrapper());
                break;
            case 8:
                boxes.push_back(box_8_wrapper());
                break;
            case 9:
                boxes.push_back(box_9_wrapper());
                break;
            default:
                break;
        }
    }
    return boxes;
}
```

```
bool Screens::determine_box_order_test() { //Ensures that the order being the pancakes are being displayed in is the correct order
```

```
  vector<int> actual = box_order_test_vector();
  vector<vector<string>> compare = box_order_test_comparison_vector();

  vector<vector<string>> boxes = determine_box_order(actual);
  bool box_order = true;
  for(int i = 0; i < (int)boxes.size(); i++) {
     for(int j = 0; j < (int)boxes[0].size(); j++) {
        if(boxes[i][j] != compare[i][j]) {
           box_order = false;
        }
     }
  }
  return box_order;
}


vector<vector<string>> Screens::box_order_test_comparison_vector() { //Provides a test pancakes stack to compare to
  vector<vector<string>> compare;
  compare.push_back(box_1_wrapper());
  compare.push_back(box_2_wrapper());
  compare.push_back(box_3_wrapper());
  compare.push_back(box_4_wrapper());

  return compare;
}


vector<int> Screens::box_order_test_vector() {//Provides a vector that should yield the same order of pancakes as the function above.
  vector<int> actual;
  actual.push_back(1);
```

```
    actual.push_back(2);
    actual.push_back(3);
    actual.push_back(4);

    return actual;
}
// end of part
5==========
==========
```

Engine.h

```
#ifndef
ENGINE_H
        #define ENGINE_H

        #include <string>
        #include <vector>
        #include <tuple>
        #include <ncurses.h>
        #include <string>
        #include "aiTree.h"
        #include "find_solution.h"

        using namespace std;

        struct Engine{
          private:
          vector<tuple<string, int, int>> high_scores;
          vector<int> user_order;
```

```
string initials;
Pancake* user_stack;
Pancake* ai_stack;
int num_moves;
int curr_difficulty;
int num_pancakes;
int winner; // 0 - player, 1 - ai , 2 tie

public:
Engine();
~Engine();
bool game_over();
bool test_game_over();
void set_order(vector<int> permutation);
void turn(int user_move);
void exec_user_move(int user_move);
int get_ai_move();
void exec_ai_move();
bool check_is_over();
bool check_user_stack();
bool check_ai_stack();
int calculate_score(int pancakes);
void flip_user(int pos);
void flip_ai(int pos);

//Test get set functions

void set_user_order(vector<int> _user_order);
vector<int> get_user_order();
```

```
    vector<int> get_ai_order();


    // return the size of a pancake stack for the current game
    int get_num_pancakes();
    // return the selected difficulty of the current game
    int get_difficulty();
    // return the player's initials
    string get_initials();
    // return the current top 5 scores
    vector<tuple<string, int, int>> get_top_5();
    // set the number of pancakes in a stack for the current game
    void set_num_pancakes(int pancakes);
    // set the difficulty of the current game
    void set_difficulty(int d);
    // set the player's initials
    void set_initials(string i);
    // set the high_scores vector (use after adding a)
    void set_high_scores(vector<tuple<string, int, int>> v);
    // give the index of the best next flip based on the current stack
    int get_hint();
    // get the minimum number of moves to solve given the current stack
    int get_min_moves();
    // return a message displaying who won
    string get_winner();
};


#endif
```

Engine.cpp

```cpp
#include "Engine.h"

using namespace std;

Engine::Engine(){
  initials = "";
  user_stack = nullptr;
  ai_stack = nullptr;
  num_moves = 0;
  curr_difficulty = 0;
  num_pancakes = 0;
  winner = -1; // 0 - player, 1 - ai , 2 tie
}

Engine::~Engine(){
  delete user_stack;
  delete ai_stack;
}

void Engine::flip_user(int pos){
  user_stack->flip(pos);
}
void Engine::flip_ai(int pos){
  ai_stack->flip(pos);
}
bool Engine::test_game_over(){
  bool all_passed = true;
  vector<int> not_sorted;
```

```
    not_sorted.push_back(4);
    not_sorted.push_back(2);
    not_sorted.push_back(3);
    not_sorted.push_back(1);
    vector<int> sorted;
    for(int i = 0; i < 4; i++)
        sorted.push_back(i+1);
    Pancake* sorted_pancake = new Pancake(sorted);
    Pancake* not_sorted_pancake = new Pancake(not_sorted);
    user_stack = not_sorted_pancake;
    ai_stack = not_sorted_pancake;
    if(game_over()) all_passed = false;
    user_stack = sorted_pancake;
    if(!game_over()) all_passed = false;
    ai_stack = sorted_pancake;
    if(!game_over()) all_passed = false;
    user_stack = not_sorted_pancake;
    if(!game_over()) all_passed = false;
    delete sorted_pancake;
    delete not_sorted_pancake;
    return all_passed;
}


bool Engine::game_over(){
    return check_is_over();
}
void Engine::set_order(vector<int> permutation){
    user_stack = new Pancake(permutation);
    ai_stack = new Pancake(permutation);
}
```

```
void Engine::exec_user_move(int user_move){
  user_stack->flip(user_move);
}


int Engine::get_ai_move(){
  ai_tree* ai1 = new ai_tree(*ai_stack, curr_difficulty);
  int ai_move = ai1->get_next_move();
  delete ai1;
  return ai_move;
}


void Engine::exec_ai_move(){
  int ai_flip = get_ai_move();
  ai_stack->flip(ai_flip);
}


bool Engine::check_is_over(){
  if(check_user_stack()){
     if(check_ai_stack()){
        winner = 2;
     }
     else{
        winner = 0;
     }
     return true;
  }
  else if(check_ai_stack()){
```

```
        winner = 1;
        return true;
    }
    else{
        return false;
    }
}

bool Engine::check_user_stack(){
    return user_stack->is_sorted();
}

bool Engine::check_ai_stack(){
    return ai_stack->is_sorted();
}

int Engine::calculate_score(int num_pancakes) {
    if(winner == 1){
        return num_pancakes;
    }
    else if(winner == 2){
        return num_pancakes*(curr_difficulty+1);
    }
    else if(winner == 0){
        return 2*num_pancakes*(curr_difficulty + 1);
    }
    return -1;
}
```

```
void Engine::set_user_order(vector<int> _user_order){//temp -alex
  user_order = _user_order;
}
vector<int> Engine::get_user_order(){//temp -alex
  return user_stack->pStack;
}
vector<int> Engine::get_ai_order(){
  return ai_stack->pStack;
}


int Engine::get_num_pancakes() {
  return num_pancakes;
}


int Engine::get_difficulty() {
  return curr_difficulty;
}


string Engine::get_initials() {
  return initials;
}


vector<tuple<string, int, int>> Engine::get_top_5() {
  return high_scores;
}
```

```
void Engine::set_num_pancakes(int pancakes) {
  num_pancakes = pancakes;
}

void Engine::set_difficulty(int d) {
  curr_difficulty = d;
}

void Engine::set_initials(string i) {
  initials = i;
}

void Engine::set_high_scores(vector<tuple<string, int, int>> v) {
  high_scores = v;
}

int Engine::get_hint(){
  vector<int>* ans = find_solution(user_stack->pStack);
  int next = ans->at(0);
  delete ans;
  return next;;
}

int Engine::get_min_moves() {
  vector<int>* ans = find_solution(user_stack->pStack);
  int min_moves = ans->size();
  delete ans;
```

```
    return min_moves;
  }


  string Engine::get_winner() {
    if (winner == 0)
        return "You beat the computer!!!";
    if (winner == 1)
        return "You lost...";
    return "It was a tie!";
  }
```

Pancake.h

```
#ifndef
PANCAKE_H
                #define PANCAKE_H


                #include <vector>
                #include <iostream>


                using namespace std;


                struct Pancake {
                  vector<int> pStack;


                  Pancake() {} // strictly for testing
                  Pancake(vector<int> stack);
                  Pancake(Pancake* parent, int childMove);
                  int at(int index);
```

```
    void flip(int flipPos);
    void test_print();
    int getSortedness();
    bool is_sorted();
    int get_stack_size();


    // tests
    bool pancake_flip_test0();
    bool pancake_flip_test1();
    bool pancake_sortedness_test0();
    bool pancake_sortedness_test1();
};


#endif
```

Pancake.cpp

```
#include
"Pancake.h
"


Pancake::Pancake(vector<int> stack): pStack(stack){} // index 0 - top



Pancake::Pancake(Pancake* parent, int childMove){
  for(int i = 0; i < (int)parent->pStack.size(); i++){
     pStack.push_back(parent->pStack.at(i));
  }
  flip(childMove);
```

```
}


int Pancake::at(int index){
  return pStack.at(index);
}


void Pancake::flip(int flipPos){ //flips the pancake at position flipPos
  if(flipPos >= (int)pStack.size()){
     cout << "Invalid flip position" << endl;
     return;
  }
  int index = 0;
  while(flipPos > index){
     int temp = pStack.at(index);
     pStack.at(index) = pStack.at(flipPos);
     pStack.at(flipPos) = temp;
     index ++;
     flipPos --;
  }
}


void Pancake::test_print(){ // not the real pancake print function
  cout << "Stack: ";
  for(int i = 0; i < (int)pStack.size(); i++){
     cout << pStack.at(i) << ", ";
  }
  cout << endl;
}
```

```
int Pancake::getSortedness(){ //get the number of reversals
  int reversals = 0;
  int comparison = -1; // 0: < , 1: >
  for(int i = 1; i < (int)pStack.size(); i++){
     int j = i -1;
     int tempComp = -1;
     if(pStack.at(j) > pStack.at(i)){
        tempComp = 1;
     }
     else{
        tempComp = 0;
     }
     if(comparison == -1){
        comparison = tempComp;
     }
     else{
        if(comparison != tempComp){
           reversals ++;
        }
        comparison = tempComp;
     }
  }
  return reversals;
}


bool Pancake::is_sorted(){
  for(int i = 0; i < (int)pStack.size(); i ++){
     if(pStack.at(i) != (i+1)){
```

```
            return false;
        }
    }
    return true;
}


int Pancake::get_stack_size(){
    return pStack.size();
}


// tests
bool Pancake::pancake_flip_test0(){
    vector<int> p_stack;
    for(int i = 0; i < 9; i++){
        p_stack.push_back(i);
    }
    Pancake p(p_stack);
    p.flip(5);
    vector<int> compare_stack;
    compare_stack.push_back(5);
    compare_stack.push_back(4);
    compare_stack.push_back(3);
    compare_stack.push_back(2);
    compare_stack.push_back(1);
    compare_stack.push_back(0);
    compare_stack.push_back(6);
    compare_stack.push_back(7);
    compare_stack.push_back(8);
    for(int i = 0; i < 9; i++){
```

```
    if(p.at(i) != compare_stack.at(i)){
        //cout << "pancake_flip_test0 failed: " << p.at(i) << " != " <<
compare_stack.at(i) << ", at index: " << i << endl;
        return false;
    }
  }
  return true;
}


bool Pancake::pancake_flip_test1(){//test on flipping one pancake
  vector<int> p_stack;
  for(int i = 0; i < 9; i++){
    p_stack.push_back(i);
  }
  vector<int> compare_stack;
  for(int i = 0; i < 9; i++){
    compare_stack.push_back(i);
  }
  Pancake p(p_stack);
  p.flip(0);
  for(int i = 0; i < 9; i++){
    if(p.at(i) != compare_stack.at(i)){
        //cout << "pancake_flip_test1 failed: " << p.at(i) << " != " <<
compare_stack.at(i) << ", at index: " << i << endl;
        return false;
    }
  }
  return true;
}
```

```
bool Pancake::pancake_sortedness_test0(){
  vector<int> p_stack;
  p_stack.push_back(1);
  p_stack.push_back(2);
  p_stack.push_back(4);
  p_stack.push_back(3);
  p_stack.push_back(5);
  p_stack.push_back(6);
  p_stack.push_back(7);
  p_stack.push_back(8);
  p_stack.push_back(9);
  Pancake p(p_stack);
  int s = p.getSortedness();
  if(s == 2){
     return true;
  }
  return false;
}


bool Pancake::pancake_sortedness_test1(){
  vector<int> p_stack;
  p_stack.push_back(1);
  p_stack.push_back(2);
  p_stack.push_back(3);
  p_stack.push_back(4);
  p_stack.push_back(5);
  p_stack.push_back(6);
  p_stack.push_back(7);
  p_stack.push_back(8);
  p_stack.push_back(9);
```

```
    Pancake p(p_stack);
    int s = p.getSortedness();
    if(s == 0){
        return true;
    }
    return false;
}
```

aiTree.h

```
#ifndef
AI_TREE_H
        #define AI_TREE_H

        #include "node.h"

        using namespace std;

        struct ai_tree{
          private:
          node* source_node = nullptr;
          int depth;
          vector<node*> leaves;
          vector<node*> id_path;
          public:
          void recursive_create(node* ptr, int lv_left);
          int get_to_be_num_children(node* ptr);

          ai_tree(){}; // strictly for testing
          ai_tree(Pancake p_stack, int difficulty);
```

```
    ~ai_tree();
    void dfs_clear(node* ptr);


    int get_depth();
    node* get_source();
    int get_next_move();
    int get_best_leaf();


    // tests
    bool get_to_be_num_children_test0();
    bool get_to_be_num_children_test1();
    bool get_to_be_num_children_test2();
    bool get_next_move_test0();
    bool get_next_move_test1();
    bool get_next_move_test2();
    bool get_next_move_test3();
};


#endif
```

aiTree.cpp

```
#include
"aiTree.h"


using namespace std;
void ai_tree::recursive_create(node* ptr, int lv_left){ // recursive algorithm for
making the tree
  int num_children = get_to_be_num_children(ptr);
  if(num_children == 0){
```

```
        leaves.push_back(ptr);
        ptr->set_level(lv_left);
        return;
    }
    if(lv_left <= 0){
        leaves.push_back(ptr);
        return;
    }
    for(int i = 1; i <= num_children; i++){
        node* temp = new node(ptr, i);
        ptr->children.push_back(temp);
        recursive_create(temp, lv_left - 1);
    }
}


int ai_tree::get_to_be_num_children(node* ptr){ // prevents unwanted children
    int num_child = ptr->get_stack()->get_stack_size();
    num_child -- ;
    int top = num_child;
    for(int i = top ; i > 0; i--){
        int max_index = -1;
        int max = -1;
        for(int j = 0; j < i+1; j++){
            if(ptr->get_stack()->at(j) > max){
                max = ptr->get_stack()->at(j);
                max_index = j;
            }
        }
        if(max_index == i){
            num_child--;
```

```
        }
        else{
            break;
        }
    }
    if(num_child < 0){
        num_child = 0;
    }
    return num_child;
}



ai_tree::ai_tree(Pancake p_stack, int difficulty){
    source_node = new node(p_stack);
    depth = difficulty;
    recursive_create(source_node, difficulty);
}


ai_tree::~ai_tree(){
    dfs_clear(source_node);
    delete source_node;
    source_node = nullptr;
}


void ai_tree::dfs_clear(node* ptr){
    if(ptr == nullptr){
        return;
    }
```

```
    for(int i = 0; i < ptr->get_children_size(); i++){
        dfs_clear(ptr->get_child_at(i));
        delete ptr->get_child_at(i);
        ptr->set_child_at(i, nullptr);
    }
}


int ai_tree::get_depth(){
    return depth;
}


node* ai_tree::get_source(){
    return source_node;
}


int ai_tree::get_next_move(){
    int best_index = get_best_leaf();
    if(best_index == -1){
        return -1;
    }
    node* ptr = leaves.at(best_index);
    while((ptr->get_parent() != source_node)&&(ptr->get_parent() != nullptr)){
        ptr = ptr->get_parent();
    }


    return ptr->get_flipped_at();
}
```

```
int ai_tree::get_best_leaf(){
   int best_sortedness = 999;
   int best_index = -1;
   int best_level = -1;


   for(int i = 0 ; i < (int)leaves.size(); i++){
      int curr_sortedness = leaves.at(i)->get_score();
      int curr_level = leaves.at(i)->get_level();


      if(curr_sortedness < best_sortedness){
         best_sortedness = curr_sortedness;
         best_index = i;
         best_level = curr_level;
      }
      else if((curr_level != -1)&&(curr_level > best_level)){
         best_sortedness = curr_sortedness;
         best_index = i;
         best_level = curr_level;
      }
   }
   if(best_index == -1){
      return -1;
   }
   return best_index;
}


// tests
```

```
bool ai_tree::get_to_be_num_children_test0(){
  vector<int> p_stack1;
  for(int i = 0; i < 9; i++){
     p_stack1.push_back(i+1);
  }
  Pancake p1(p_stack1);
  node node1(p1);
  ai_tree ai1;
  int test1 = ai1.get_to_be_num_children(&node1);
  if(test1 == 0){
     return true;
  }
  return false;
}


bool ai_tree::get_to_be_num_children_test1(){
  vector<int> p_stack1;
  p_stack1.push_back(7);
  p_stack1.push_back(6);
  p_stack1.push_back(5);
  p_stack1.push_back(4);
  p_stack1.push_back(3);
  p_stack1.push_back(2);
  p_stack1.push_back(1);
  p_stack1.push_back(8);
  p_stack1.push_back(9);
  Pancake p1(p_stack1);
  node node1(p1);
  ai_tree ai1;
  int test1 = ai1.get_to_be_num_children(&node1);
```

```
  if(test1 == 6){
     return true;
  }
  return false;
}


bool ai_tree::get_to_be_num_children_test2(){
  vector<int> p_stack1;
  p_stack1.push_back(9);
  p_stack1.push_back(8);
  p_stack1.push_back(7);
  p_stack1.push_back(6);
  p_stack1.push_back(5);
  p_stack1.push_back(4);
  p_stack1.push_back(3);
  p_stack1.push_back(2);
  p_stack1.push_back(1);
  Pancake p1(p_stack1);
  node node1(p1);
  ai_tree ai1;
  int test1 = ai1.get_to_be_num_children(&node1);
  if(test1 == 8){
     return true;
  }
  return false;
}


bool ai_tree::get_next_move_test0(){
  vector<int> p_stack1;
```

```
  p_stack1.push_back(1);
  p_stack1.push_back(2);
  p_stack1.push_back(3);
  p_stack1.push_back(4);
  p_stack1.push_back(5);
  p_stack1.push_back(6);
  p_stack1.push_back(7);
  p_stack1.push_back(8);
  p_stack1.push_back(9);
  Pancake p1(p_stack1);
  node node1(p1);
  ai_tree ai1(p1, 5);
  int next_move = ai1.get_next_move();
  if(next_move == -1){
     return true;
  }
  cout << next_move << " ";
  return false;
}


bool ai_tree::get_next_move_test1(){
  vector<int> p_stack1;
  p_stack1.push_back(8);
  p_stack1.push_back(7);
  p_stack1.push_back(6);
  p_stack1.push_back(5);
  p_stack1.push_back(4);
  p_stack1.push_back(3);
  p_stack1.push_back(2);
  p_stack1.push_back(1);
```

```
    p_stack1.push_back(9);
    Pancake p1(p_stack1);
    node node1(p1);
    ai_tree ai1(p1, 5);
    int next_move = ai1.get_next_move();
    if(next_move == 7){
        return true;
    }
    cout << next_move << " ";
    return false;
}


bool ai_tree::get_next_move_test2(){
    vector<int> p_stack1;
    p_stack1.push_back(2);
    p_stack1.push_back(3);
    p_stack1.push_back(4);
    p_stack1.push_back(5);
    p_stack1.push_back(6);
    p_stack1.push_back(7);
    p_stack1.push_back(8);
    p_stack1.push_back(1);
    p_stack1.push_back(9);
    Pancake p1(p_stack1);
    node node1(p1);
    ai_tree ai1(p1, 5);
    int next_move = ai1.get_next_move();
    if(next_move == 6){
        return true;
    }
```

```
    cout << next_move << " ";
    return false;
}

bool ai_tree::get_next_move_test3(){
    vector<int> p_stack1;
    p_stack1.push_back(9);
    p_stack1.push_back(8);
    p_stack1.push_back(7);
    p_stack1.push_back(6);
    p_stack1.push_back(5);
    p_stack1.push_back(4);
    p_stack1.push_back(3);
    p_stack1.push_back(2);
    p_stack1.push_back(1);
    Pancake p1(p_stack1);
    node node1(p1);
    ai_tree ai1(p1, 5);
    int next_move = ai1.get_next_move();
    if(next_move == 8){
        return true;
    }
    cout << next_move << " ";
    return false;
}
```

Node.h

```
#ifndef
NODE_H
#define NODE_H
```

```
#include "Pancake.h"

using namespace std;

struct node{
  public:
  Pancake p_stack;
  int sortedness_score;
  int flipped_at;
  node* parent;
  vector<node*> children;
  int level = -1;

  node() {} // strictly for testing

  node(Pancake p_stack): p_stack(p_stack), flipped_at(-1), parent(nullptr){
     sortedness_score = p_stack.getSortedness();
  }
  node(node* p, int flip_ind);

  node* get_parent();
  int get_score();
  node* get_child_at(int i);
  void set_child_at(int i, node* s){
     children.at(i) = s;
  }
  Pancake* get_stack();
  int get_flipped_at();
```

```
    void set_level(int d);
    int get_level();
    int get_children_size(){return children.size();}

    // tests
    bool node_tester();
};

#endif
```

Node.cpp

```
#include "node.h"

using namespace std;

node::node(node* p, int flip_ind): p_stack(*(p->get_stack())){
    parent = p;
    p_stack.flip(flip_ind);
    sortedness_score = p_stack.getSortedness();
    flipped_at = flip_ind;
}

node* node::get_parent(){
    return parent;
}

int node::get_score(){
```

```
  return sortedness_score;
}


node* node::get_child_at(int i){
  if((i+1) > (int)children.size()){
      return nullptr;
  }
  return children.at(i);
}


Pancake* node::get_stack(){
  return &p_stack;
}


int node::get_flipped_at(){return flipped_at;}


void node::set_level(int d){
  level = d;
}


int node::get_level(){
  return level;
}


// tests
bool node::node_tester(){
  vector<int> p_stack;
```

```
    for(int i = 8; i >=0; i --){ //8,7,6,5,4,3,2,1,0
        p_stack.push_back(i);
    }
    Pancake p(p_stack);
    node n(p);
    if(n.sortedness_score != 0)
        return false;
    node n1(&n, 5);
    if(n1.get_score() != 1)
        return false;
    if(n1.get_flipped_at() != 5)
        return false;


    return true;
}
```

Test.cpp

```
#include "Screens.h"
#include "aiTree.h"
#include "node.h"
#include "Pancake.h"
#include <iostream>
#include <assert.h>
#include <unistd.h>
using namespace std;


int main() {
    Screens s = Screens();
```

```
cout << "Testing File Read...";
assert(s.test_file_read());
cout << "Passed" << endl;

cout << "Testing File Write...";
assert(s.test_file_write());
cout << "Passed" << endl;

cout << "Testing Score Add...";
assert(s.test_add_score());
cout << "Passed" << endl;

//cout << "Testing splash screen...";
//assert(s.test_splashscreen());
//cout << "Passed" << endl;

//cout << "Testing Random Stack Order..." << endl;
//assert(s.test_random_stack_order());
//cout << "Passed" << endl;

//cout << "Testing duplicate check" << endl;
//assert(s.test_user_stack_duplicate_check());
//cout << "Passed" << endl;

//cout << "Testing user range check" << endl;
//assert(s.test_user_stack_range_check());
//cout << "Passed" << endl;
```

```
//cout << "Testing arrow move" << endl;
//assert(s.test_arrow_move());
//cout << "Passed" << endl;


//cout << "Testing game over" << endl;
//assert(e.test_game_over());
//cout << "Passed" << endl;

/*cout << "Testing Box Order...";
assert(s.determine_box_order_test());
cout << "Passed" << endl;

cout << "Testing Center Output...";
assert(s.center_output_test());
cout << "Passed << endl;
*/

//s.test_player_selection(0,0);
//s.test_blink_stack();


//-----------------------------------------------
Pancake p = Pancake();


cout << "Testing Pancake Flip 1...";
assert(p.pancake_flip_test0());
cout << "Passed" << endl;
```

```
cout << "Testing Pancake Flip 2...";
assert(p.pancake_flip_test1());
cout << "Passed" << endl;


cout << "Testing Pancake Sortedness 1...";
assert(p.pancake_sortedness_test0());
cout << "Passed" << endl;


cout << "Testing Pancake Sortedness 2...";
assert(p.pancake_sortedness_test1());
cout << "Passed" << endl;


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

node n = node();


cout << "Testing Node operations...";
assert(n.node_tester());
cout << "Passed" << endl;


ai_tree a = ai_tree();
cout << "Testing AI get_next_move 1...";
assert(a.get_next_move_test0());
cout << "Passed" << endl;
```

```
    cout << "Testing AI get_next_move 2...";
    assert(a.get_next_move_test1());
    cout << "Passed" << endl;

    cout << "Testing AI get_next_move 3...";
    assert(a.get_next_move_test2());
    cout << "Passed" << endl;

    cout << "Testing AI get_next_move 4...";
    assert(a.get_next_move_test3());
    cout << "Passed" << endl;

    cout << "Testing AI get_to_be_num_children 1...";
    assert(a.get_to_be_num_children_test0());
    cout << "Passed" << endl;

    cout << "Testing AI get_to_be_num_children 2...";
    assert(a.get_to_be_num_children_test1());
    cout << "Passed" << endl;

    cout << "Testing AI get_to_be_num_children 3...";
    assert(a.get_to_be_num_children_test2());
    cout << "Passed" << endl;

    cout << "All Tests Passed!" << endl;
}
```

Section 11 - Bibliography

[1] “GNU Make.” *GNU Make*, 22 May 2016, www.gnu.org/software/make/.