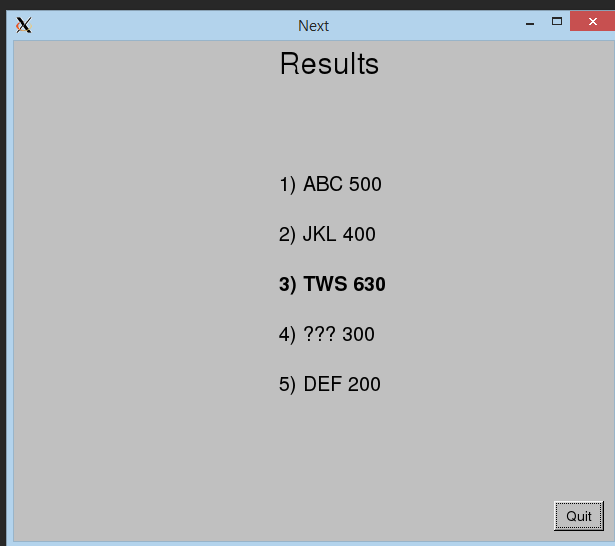
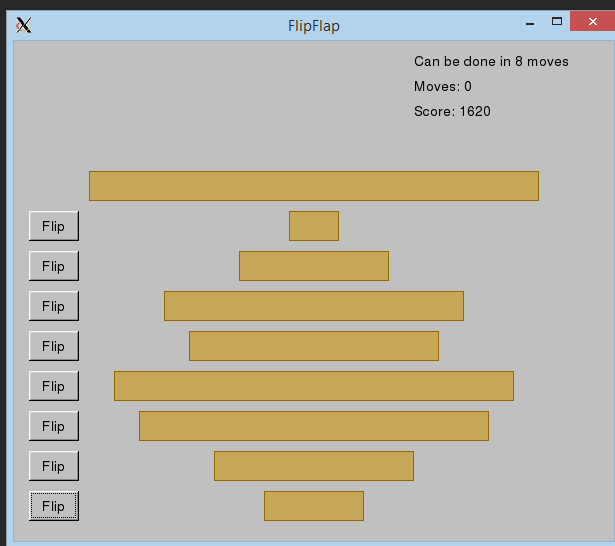
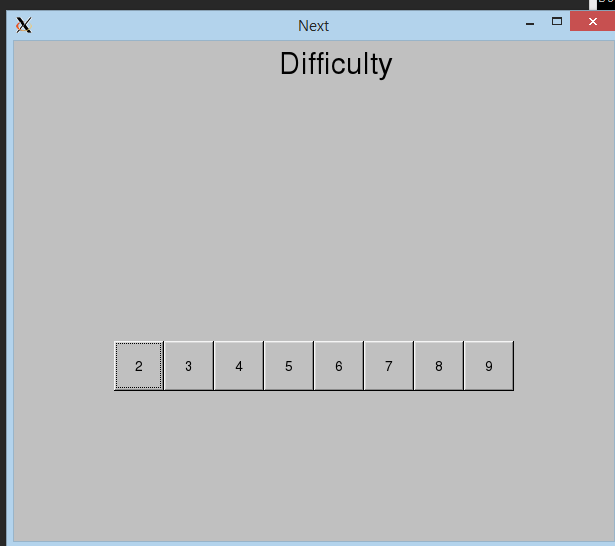
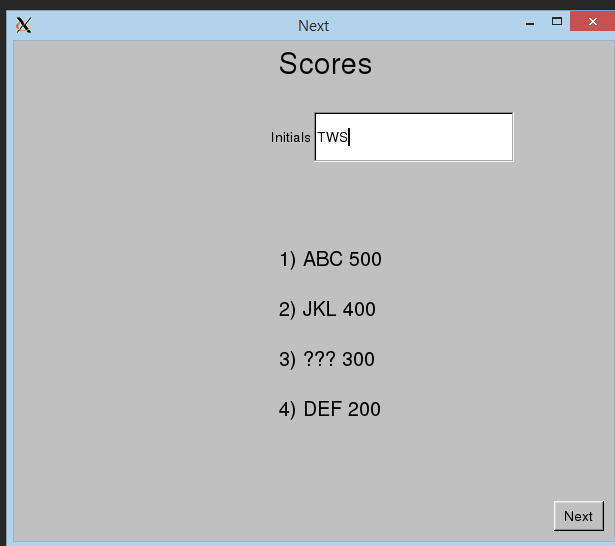
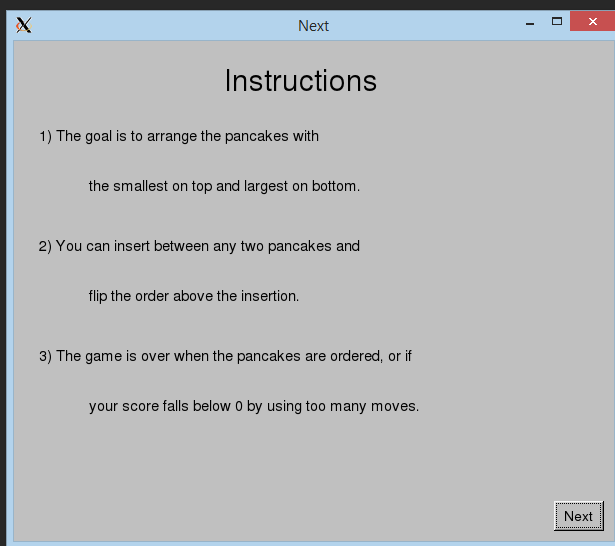
The members of team 37 include Travis Stebbins ‘18, Sam Stegall ‘18, and Allison Snodgrass ’17. Sam was responsible for giving the user instructions on how to play FlipFlaps, calculating the scores, and displaying them once the game concluded, while Allison was responsible for the splash screen and the displaying the pancakes randomly on the screen. Travis worked on asking for the player’s initials and displaying the top scores, along with allowing the user to flip the pancakes. Buttons were implemented to flip the pancakes. Travis also edited the individual codes to work cohesively and make the game run smoothly, this included buttons that allowed the user to go to the next window.

FlipFlaps may seem like a silly, fun game, but it demonstrates a complex mathematical problem. The pancake problem begs the question of the flips needed to sort elements of an arbitrary permutation, or in other words, the number of flips needed to put a stack of unequally sized items in order from smallest to largest. It has been proven that at least 2n-3 flips are necessary to solve the problem. Bill Gates proved that (5n+5)/3 flips will always be enough to solve the problem, and the problem shows significance in multiple fields. “In general, a pancake stack is an example of a data structure, and the pancake problem is relevant, for example, to the construction of networks of parallel processors, in which pancake sorting can provide an effective routing algorithm between processors. Pancake sorting also provides insights into evolutionary processes. In any evolutionary process, changes in DNA sequences (genomes) can cause a new species to split off from an existing one, thus leading to the diversity of life-forms that we know today.”( Peterson, Ivars. "Ivars Peterson's MathTrek." Web. http://www.uwyo.edu/bshader/mathtlcalgebra/pancake.pdf.)

While given ample time to finish the project, there will always be small details that deserve more attention, so time was the major limitation during the project; going along with that, trying to find time to meet was difficult. There were some ideas we wished we’d been able to add to the project that we were unsure how to implement, such as a countdown timer and the user visually seeing the pancakes flip. Again, though, given more time for research these would have been accomplishable.

The main file, game.cpp, creates windows and connects them all using a few public functions. Game.cpp also attaches a "Next" button to each necessary window that hides the current window, shows the next window, and calls any other necessary functions when the button is pushed. Each window necessary for the game to run is a struct that extends the Window class and has its own .header and c++ file, if necessary. The splash screen and instructions are simply created with graphical objects. Scores\_window displays all of the current top 5 high scores (stored in scores.txt) and has an In\_box that lets the user input their initials. The initials are called when the “Next” button in Scores\_window is pushed and are then returned by a public function; if the user enters more than 3 letters, only the first 3 will be returned. Initials are paired with scores, read in from the file, and text objects are made to display the scores on the screen.

The game is played on Game\_window, which implements the Stack and Pancake classes. The Pancake class simply has a width variable, and the Stack class contains a vector of pancakes and functions to modify it. The create\_stack function adds the number of pancakes equal to the difficulty level and then shuffles them in a random order. The flip function flips the stack of pancakes based on an input. In the Game\_window class, the constructor draws rectangles for each pancake based on the pancake's width and also creates and attaches the number of buttons necessary to flip the number of pancakes that there are. The flip function in Game\_window calls the flip function on the Stack class and then calls the redraw\_game\_screen function, which is the main function for Game\_window. redraw\_game\_screen attaches a gray rectangle on top of the existing pancakes to cover them and then redraws the pancakes based on their new order from the flip function. Then, it checks to see if the player has beat the game by putting the pancakes in the right order or if the score is less than 0; in either case, the game is ended. Lastly, the Result\_window class displays the scores just like the Scores\_window (although without the In\_box), and it also displays the player's score in bold. Then, it adds the player's score to the vector of high\_scores, sorts it from highest to lowest, erases the lowest score so there are only 5 again, and outputs the scores to scores.txt again. 

Overall, we feel that our project has been very successful. The game does everything that is asked of it and more, and the code is simple and incredibly well organized. There were certain aspects, such as the use of permutations, that were deemed too lengthy and complicated in comparison to the alternatives; in the case of permutations this would be the use of the shuffle function. One of our main goals was to keep the code as simple as possible, and we feel we have achieved that. The game runs without a hitch and is very user friendly. The use of buttons makes the game seem more realistic and less like a project.

Through multiple times of playing the game, one would hope the user would start to recognize there is an algorithm for solving the stack in the quickest way possible. This would show that math is an instrumental resource in playing this game. While the game does not show that the pancake problem can correlate to anything other than pancakes, it is interesting to learn that there is real scientific research developing through this. Given any research at all, a user would learn the same things we did: the problem is not just a silly game, but an applicable problem to many situations. We learned some interesting coding techniques as well, purely through research and the aspiration to make the project as efficient and functioning as possible.

While the code is exactly what we would want for what the program does, there are many things we wanted to do if there had been more time. We would’ve had music playing throughout the game, had the game windows match the splash screen, and made the pancakes look more like pancakes. We could’ve made the game more visually pleasing, and in turn, game-like. We also wish we’d implemented a countdown timer, so the user only had a certain amount of time to beat the game. It would've been very cool to watch the pancakes flip. There were many ways the game could’ve improved, but the code was very nicely thought out. Perhaps there were more simple ways to implement some of the processes our program performs, but we put together a program that is easy to follow.

The program is very simple to run. All of the header and c++ files of ours must be in a folder with the header and c++ files of fltk. Using the command "g++ -std=c++14 \*cpp \*o -lfltk -lfltk\_images” in the terminal to compile. To run, type “./a.out” and enter. This will bring up the splash screen, and the user will click through the windows in order to play the game.

Because there were multiple small codes used and the code was so simplistic, there were very few comments needed. Comments were used solely to label what a block of code was, for example: functions, objects, constructors, variables, and constants.