Planning:

This project really forced me to plan out what I was going to do and how I was going to do it.

I realized that I shouldn't be going through the motions when coding and I need to realize that there are more options than what I first think of.

I felt like the pre-planning for this assignment was the most essential part because there was a lot that could go wrong if did not have a clear idea of what you were doing.

Creating a clear plan for the program before even creating your file is extremely important.

I really honed in my ability to take a problem to a whiteboard and come back to my computer with a solution.

I learned that thoroughly planning your code before beginning is crucial to the performance of the code and minimizes time spent actually writing code.

Also, when I code, I need to take some time to think of what exactly I am trying to do, what function I am trying to create, and I need to think of the simplest possible way I can create this function. If I just would take a little time to think about the code, I think I would solve a lot of my problems regarding debugging frustration.

I first learned how important planning out your code before writing it out is to successfully implementing long, complicated algorithms.

If there's anything that I learned when completing this assignment, it can be easily summed up in two parts. Firstly, if you don't plan out your code before you start, it's not going to go well. Secondly, if you don't organize your code so you can follow what everything is supposed to do, it's not going to go well.

Something I learned about programming from this assignment is how much writing pseudocode can help before starting to code.

One thing I learned about programming from this assignment was that when you undertake a project, there can be many blind alleys that you can go down that lead you very close to the answer but not actually to the answer, leading a lot of wasted time.

I also learned that you really have no guarantee of finishing a coding project before a certain time; you can put in many hours into finishing the assignment, but it isn't like projects in other classes where you are guaranteed to finish if you put in enough time to do the research and write the paper or make the presentation.

I learned that planning out my code beforehand and not mindlessly coding is most definitely the ideal way to write smart code.

Generally for assignments I would go straight into coding and do little planning, but with this assignment, of the work I was able to do, I found myself methodically working out strategies I could use to solve each of the parts. I learned that it can be beneficial to take extended periods of time planning and settling on an idea before rushing into coding.

I need to make sure before I undertake a problem/lab like this that I understand all the special cases or important things about a problem.

I found that if I planned each step in the code (brainstorming, writing on paper, coding, debugging) immediately when we received the assignment, and actually following through and staying on schedule, greatly reduced how stressful and overwhelming the assignment was, and actually made the assignment more enjoyable.

In future labs, I should spend time brainstorming prior to the lab, especially in terms of coming up with a sketch of the algorithm and a list of helper methods I should code.

I also learned that is when planning out algorithms or pseudocode, it is important to make the plan detailed.

I have always been guilty of just making small changes and running the program numerous time without taking time to think out the whole process, but this mindset was drastically changed after this lab.

I think the biggest thing I took away from this is that every complicated Computer Science problem can be broken down into simpler ones that can then be combined to solve the big problem.

I learned that I need to plan out my code logically more than what I do now.

I didn't spend enough time planning out the lab, took way too much time stuck on trying to figure out a complex algorithm, and left way too much work for the last week in which the lab was due.

I learned the importance of planning my code beforehand from this assignment.

One major lesson that I learned from this assignment is the importance of planning.

I realized that one of the reasons I was actually having trouble starting the assignment is because I didn't know where to start, so breaking down the tasks before I start coding would be beneficial.

I didn't realize until doing this assignment that knowing syntax is not going to be beneficial if I don't know how to solve the problem at hand.

A way I can prevent this mistake from occurring again is, along with breaking down the tasks, is to establish a sort of pseudo-code for each "section."

It is important to plan out the general stages of the project.

I found it really interesting how effective breaking down a problem into small problems would be. This allowed my to have a clear train of thought throughout the entire coding process that I haven't really had before. I knew exactly what I was doing and what I needed to do, even with such an open ended lab.

One big thing that I learned from this assignment is that detailed planning (more than just method names) is needed for assignments like these. Although I did have a decent call tree and an outline of what each method should take as inputs and return, I leaped from naming the methods to writing the methods with the actual manner in which the methods would function still foggy in my head. This resulted in a situation where after a few hours of coding, I had a program that made my CPU regret ever being made. The first time a 'completed' crossword appeared on my screen it took about a minute and was a grid of 16 E's. Because I spent no time planning how the methods would work, they were slow and wrong. These were all eventually scrapped after I sat down with a pencil and paper and mapped out specifically how the main methods would function almost line to line.

The most important thing about programming I learned from this project is the ability to break down a complex task into many small, more manageable, sections.

I thought about different ways to approach it before settling on a specific algorithm. Before writing out any methods, I wrote down this algorithm with specific method names so that I knew how everything would fit together before I began to code. These extra couple hours that I spent coming up with my structure before coding ended up being extremely beneficial to me and led to probably my most efficient hour and a half of coding of my life. (…) In addition, it made debugging easier because I knew exactly how my methods worked (or were supposed to work) and could more easily find the problem in my code.

For such a large assignment, it is better to plan everything on paper beforehand and not plan as you go.

I realized the importance of planning out what I would code before I would actually start coding.

I found it easier to divide the lab into smaller components and aim to complete a set number of methods a day. Even though I did not always reach my goal, my planning enabled me to feel less stress in regards to the lab because I had a clear defined path forwards.

I think that writing down my pseudocode would help with reading the code, understanding how everything that I did works, and which specific methods or algorithms have apparent flaws or alternative solutions that result in greater efficiency.

I think crossword definitely taught me that I need to start paying more attention to what I write, thinking how I can translate one part of my code to another, and evaluating what they ask the code to do with their inputs through reading it.

Another thing I learned is that I probably should not code right off the bat without developing a good understanding of the problem or coming up with working pseudocode.

One thing I learned about programming while doing this assignment was that it is important to approach problems from different ways.

I began to write down everything I wanted to accomplish in each part of the assignment and then I tried to sort them into methods. So basically, I broke the assignment in half and then I broke those pieces into smaller problems until I felt like I was ready to begin programming. This made the assignment seem more approachable and I hope to do this very every assignment afterwards.

I only really took into account the time I would spend on the first idea I had, not considering the possibility that it would fail.

I learned that I am very eager to make quick changes to code in order to get to the end goal, without considering the further implications.

Through this assignment, I initially learned the great benefits of framing out the algorithm in a piecemeal fashion on paper first before actually coding.

This problem taught me the importance of proper planning.

Although I've always known that planning was important, it has always been too easy to add another argument to a method or copy paste one section of code somewhere else so this warning was often something I brushed off. This was my downfall for this lab.

I should have spent more of my off time just thinking about my code and mentally planning it out, taking notes on my phone if I got any ideas.

While doing this lab, I did not plan well. I simply dove into coding, which was the wrong step.

One thing I learnt about programming is that it's important to just simply take a second to step back, reflect on the state of the code at the moment, and reassess whether the strategy actually makes sense or not.

I need to have a plan going into a large project, not just winging it the entire time.

Additionally, writing out pseudo-code by hand in the form of flow charts helped me visualize my algorithm for placing words. This allowed me to understand what my code was doing and printing out board states along the way while working out test cases on paper really allowed me to understand what my code was doing so I could fix and improve it.

I learned that I can code more efficiently after making an outline, even if I already understand the purpose of all the methods I need to code.

This shows that I should not necessarily implement whatever comes to mind first when planning out an assignment; rather, I should think of several possibilities and choose based on weighing factors such as efficiency in runtime and efficiency in actually writing the code.

I learned from Crossword that programming requires a large amount of planning and organization.

Formatting / clean code:

Another thing I learned is that I should pick method names more wisely.

I learned that it is always good to comment some on your code so you can more easily navigate and understand it.

Something else I learned from this assignment was the benefits of cleaning up messy code.

Organizing code makes the debugging process go much more quickly. I found that having clean code helped me be more efficient and less frustrated.

My code's readability was an issue that plagued me throughout the duration of the crossword experience.

In future assignments, I plan to better break up my code, rid the file of extraneous lines, and maybe even add comments without being prompted.

I realized that in large assignments like Crossword my variable names are just absolutely terrible and confusing.

In a longer and more complicated assignment like this, being able to read your code and keep it neat turned out to be essential for debugging and finding mistakes that existed in my logic.

For one thing I make sure to comment the important parts of my code with at least basic descriptions of what said block of code does. This is useful for when I come back to my code after taking a break or when I get lost.

I also learned that I am not very good at coming up with variable names. (…) This was especially a problem in this lab because I did this lab over the span of a couple weeks and would have to modify certain parts after not looking at it for about a week, so having better variable names honestly would have saved me some time.

Another skill I re-realized the value of is commenting code and making notes.

One of the number one thing I felt I learned about is top down programming and of creating clean, organized code.

I realize that the more organized my code, the better.

When in the middle of coding, I always just named variables and methods the first things that came to mind, so I often forgot exactly what a variable actually was or what a function was actually supposed to do. I also failed to ever comment, meaning that I had to trace, figure out, and attempt to remember how my convoluted, many-lined functions worked.

Debugging:

Through this assignment, I was able to learn more about efficient debugging methods by using print statements to see exactly where my code was not performing how I wanted it to.

Something that I learned during this assignment was how to use the debugger in pycharm.

I (out of necessity) got a lot better at debugging during this project.

Another thing I deduced about programming was that debugging is built into the foundations of programming.

I also learned some interesting other skills, such as how to use colorama to nicely format print statements.

Through the Crossword assignment, I was able to learn more about how to debug a malfunctioning program.

I formed a better relationship with the PyCharm debugger and increased the efficiency with which I shifted between running a program with and without the PyCharm terminal.

I developed better intuition for how and when to use breakpoints and how to determine when a case of interest had been reached.

I learned the importance of debugging my code without excessive print statements, particularly through the use of the PyCharm debugger.

I realized that debugging along the way helps a lot and makes figuring out issues much faster.

For debugging, I learned that printing things is a lot more quick and efficient than using the debugger tool.

Another aspect I learned about programming is that debugging is necessary for when you made a simple coding or logical mistake even when it seems tedious.

The use of debugging in the lab empowered me to better analyze my code.

By paying attention to the specific test cases and reading through my code line by line, I could speed up the development of my code and find ways to enhance the efficiency of my algorithms.

I say that this assignment helped me learn both how to debug effectively and how to increase the efficiency of methods and programs.

I definitely learned a lot about debugging correctly and doing it well.

For this assignment, I printed out variable values at select steps and placed breakpoints more strategically which helped me save time and find the error most of the time.

It was common to receive cryptic errors that did not effectively communicate the problems in your code. As a result it was essential to have an effective method of debugging, that allowed the programmer to systematically step through their code and determine the errors. One method I achieved this, was by simplifying my test cases into ones that could be easily debugged.

I learned that I should test small pieces of code before testing my full program or a large block of code.

One particular skill that I think I improved the most on was my debugging skills.

Using the red debugger dot was not very time efficient and I quickly learned this after having to click through many recursive calls until I got to the depth where an error occurred. Instead, I learned to better utilize print statements for debugging purposes.

I learned new strategies for debugging using detailed print statements.

What I learned to do to debug more efficiently was, firstly, to block off sections of my code and test parts individually.

Finally, although this seems very obvious and intuitive, I learned of the extreme importance of debugging print statements.

From this assignment I feel I have become much better at debugging.

I found out that a better way was to use print statements to either check if my code was ever reaching that part of the code before erroring out or to print out the variables to see what was wrong with them.

This assignment taught me just how much time needs to be set aside for debugging.

PyCharm's built-in debugger was little help in tracing the convoluted recursive methods that I built.

I learned that whenever I copy and paste, it's crucial that I review each line to make sure I haven't made an error, which seems tedious but would save huge amounts of time later on if the code was to bug.

Modularizing code:

I also learned how much easier it was to separate methods that accomplish different tasks.

I think one of the most important things I learned how to do was divide everything up into methods first that allowed me to identify things and then also methods that I could use over and over again.

I broke up my code into many smaller functions so that it would be easier to debug and I tested each smaller function independently so I knew it worked before putting everything together. This made it much easier for me to debug because I would only have one small method to debug versus a large method that I would have no idea where to even begin with.

Through this assignment I learned about the value of helper methods.

I got more experience with learning how to break up my code into logical pieces and methods.

When I code, I need to make sure to use top-down programming and break up and delegate tasks into smaller helper methods. Every time I create these helper methods or small functions, I need to immediately debug and assure they work properly before I move on, because if I don't, later on I won't be able to easily determine the fix.

For this assignment I made an extra effort to break down my logic into smaller methods. This proved to be very pleasant because instead of debugging many tasks in one method, I was able to pinpoint where exactly some problems of my code were.

The main thing I learned about programming during the crosswords lab was that refactoring after each big step in code is very, very useful. (…) One of the most useful parts of refactoring for me was splitting code into functions.

The number one thing I learned while going through this project was the importance of creating methods for subproblems.

The most important thing I learned how to do was debug effectively. (…) I decided to split up some of my methods into combinations of even more methods, so that I could test each method individually rather than try and guess the error in a relatively large method.

I created several global data structures and variables that I used throughout a majority of my methods as well as creating several helper methods to make my code more readable.

On this lab, I realized that splitting up my code into multiple smaller methods would make my code much easier to read.

Overall, I also maintained a well-organized piece of code, breaking up sections into methods, making the code very modular and easy to read.

Another debugging strategy I used was the creation of additional helper methods to test my code.

I also learned to compartmentalize my code into blocks that are easily understandable.

I like to make as many helper methods that I think I need. Before I move on to making the next helper method though, I make sure that the one that I just finished works properly and test it using a variety of test cases that either I made up or was provided to me. This gives me insight to what is properly working and what is not, and most of my helper methods were not completely correct after first coding them up.

Modularization is a must when coding.

I also realized the value in creating smaller helper methods because when you are creating huge methods like when I tried to do blocking all in a single method, and there was an issue that you just can't figure out.

Another thing I learned is to check each method as I went for debugging. Usually, I would write a couple of methods in a row without checking them. In this program, since methods depend heavily on each other, it is best to check each function as I write it.

Another thing I learned throughout the revisions and attempts at blocking was that it is important that I create smaller methods to tackle smaller problems and bugs so they are easier to pinpoint and fix.

I broke up longer methods into smaller methods which the main method could call. Through this process, I was able to focus on optimizing small sections of code at a time and using the profiler, I could see how each change affected my program. (…) I would look at the method with the greatest total time to see I could optimize it. I nothing immediately caught my eye, I would break up the method in to multiple smaller methods and run the program again.

One of the things I learned was the importance of splitting functions. Well-written code can be traced, and easily(?) understood. That typically can't be done with extremely long and convoluted methods. By splitting them and naming them logically, it is easier to follow the programmer's train of thought.

That strategy of breaking down methods really helped break up the problem and make more progress.

I usually just try to code everything in one big method, which leaves it a mess to debug and read. Breaking my code into smaller modules would make it so that it's a lot cleaner.

The lab had many parts to it, requiring me to requiring me to plan out what tasks I wanted to divide up into methods. (…) By organizing my code into small methods, my larger methods were much more clear to understand because reading through the method names sequentially was almost like reading comments of what the code was doing.

I definitely realized that the more methods I broke my code into, the more easily I was able to write and debug my code.

I learned how to identify ways to break down each large section into methods.

I learned that doing everything in one continuous chain is not a cool guy move, and doesn't really work for hard assignments like this one, and by extension, that breaking things into smaller tasks is not lame.

I feel that the Crossword assignment has increased my programming skills at two tasks – dividing up assignments into easier to write methods and components, and creating heuristics that use fail conditions to create more optimum responses.

Through this assignment, I learned how to effectively break down tasks into subtasks so that my code could be much easier to understand and could employ recursion.

After numerous attempts at coding the assignment and restarting from scratch, I eventually decided to plan out the structure and essential methods of my code via a top-down programming approach.

Recursion:

Above all, what I gained most from Crosswords was losing an aversion to recursion. Rather than favoring one type of algorithm over the other, I made iteration and recursion equal tools when faced with a new challenge.

I believe the primary coding skill that I improved on through Crossword was using data structures to assist in messy recursion.

Always have your Base cases in the beginning so that they are clear. When you are running a loop within a recursive algorithm, make sure your loop does not eventually get to bad input.

I discovered that I like it most when variables containing as much information as possible about the current recursion are passed as arguments and returned from recursive calls.

This lab helped me get better at writing recursive code and understand how different pieces in a recursive method fit together.

I also learned a lot more about tracking complicated recursive calls and how to backtrack through coding this lab.

This lab forced me to completely understand the power of recursion. I also learned how to make my recursive methods cleaner and easier to read and understand.

I learned how to use the idea of recursion and incorporate it to create a more compact and efficient code structure.

I also learned that recursive code should be well planned and it is almost necessary to formulate an algorithm beforehand.

Overall, I felt as if I learned more about how recursion actually worked more in depth and how to formulate clean, recursive code.

Another major part of this project was learning how to write better recursive methods. (…) I learned that I just needed to write it more cleanly by first starting with all of the base cases that either fail or work.

Global Variables

I learned how to use global variables.

Another more specific thing that I learned after finishing the assignment is that I should, as gnasher729 on stack exchange puts it, 'avoid global variables like the plague'.

Efficiency:

I got practice trying to cut time anywhere I could for programming efficiency.

The biggest thing I can say I learned after this lab was the power of heuristics. I was unable to solve some test cases due to the randomness in my program, and I feel if I implemented heuristics that made my code more optimized, I would have been able to pass all the test cases.

As for the applied techniques, in this lab I used string manipulations (learned from other labs), depth-first search (for blocking), csp (for filling words), standard recursion (for filling in sectioned off blocking areas), and regex (for finding words).

Another skill about programming that I learned is the need for efficiency. (…) This manifested in avoiding unnecessary loops and copies.

This exploration of coding techniques allowed me to understand what contributes to slowing down your run time and decreasing the performance of your code, especially in a recursive setting. (…) First, I often found myself instantiating lists in recursive methods every single time I called the method. This was a waste since I could just instantiate these lists once and use them multiple times instead of having to create one every time. Similar to this, I realized that many of my calculations were nested within loops. This meant I was making the same calculation over and over, when I could really just do it once and be done with it. Simple fixes like this one improved efficiency significantly.

I also learned how to better incorporate the concepts and search algorithms we had learned in class to solve another problem I had not ever thought about.

Over the course of the lab I figured out ways in order to decrease the effective O(n) complexity of your code through instituting checks that will catch your errors quicker.

Regex:

I learned about the relative inefficiency in time that regex had.

What I learned on the coding side from this project is that regex takes extremely long.

This helped me fully understand the applications of regular expressions.

I learned how to implement proper Regex tools and functions that were very beneficial in my code and thus, I realized how useful they are.

I learned how to better utilize and leverage regular expressions in this project.

Using set intersection instead of regexing the dictionary was such a dramatic increase in speed.

We learned how to implement regex into python programs, as opposed to using them by themselves in the grader.

Python uses a C implementation of regex, so calling regex multiple times takes up a lot of time, due to the code having to continually compile the regex to C.

I learned how to implement proper regex tools in python and functions that were very beneficial in locating the words in my puzzle, through which I learned how useful regex is in AI projects involving for example semantics or natural language processing.

I found the application of regular expressions in this lab to be extremely interesting. Although I only needed to use regular expressions to update my dictionaries, they were uniquely useful in this case.

Python features:

I also learned some functions, such as the Duplicate() function that I never used before.

I find myself strangely attached to the little tools and gimmicks Python implements, such as using an empty iterable or a modulus function as a condition for a boolean, since 0 and False are equivalent in Python. There are a plethora of cool tricks to use in python, and I'm excited to discover more cool functions like zip() and enumerate().

I also learned cool pythonic tools such as lambda, enumerate, zip, ect.

PyCharm Features:

I figured out that I could input parameters into my project configurations so I didn't have to run my code from the terminal every time and I could easily change my parameters as needed.

Specific to PyCharm, I discovered that instead of entered arguments into the command line, you could set a configuration in the settings and run with those arguments.

This is assignment was the first time I began defining methods within other methods. This allowed for better organization of my code, and made it easier for me to view and focus on specific sections of code since pyCharm can collapse methods by indentation.

Eventually, I learned that I could change the parameters in the configuration in pycharm, which allowed me to use sys.argv even when debugging, so I didn't have to switch back and forth between that and the string in the code.

Choice of data structures:

Data structures are a huge part of AI, and not only creating them to have useful data but also building them are large parts of the efficiency of your code.

The most important programming aspect I learned was thinking through which data structures to use before starting to code.

Profiler:

From this lab, I learned the usefulness of running code with a profiler.

I got better at using the profiler and analyzing the results. This definitely helped me when I was trying to make my code more efficient.

I learned about profilers, which provide statistics that describe how often and for how long methods in the program are called.

The profiler also turned out to be immensely helpful in finding the most inefficient parts of my code. This would seem like a no-brainer, but what was interesting and helpful was that the profiler actually gave me very different numbers when I changed the puzzle size.

Another tool I learnt to use because of this assignment was the built-in profiler for Pycharm.

Getting help:

The importance of talking an algorithm through with someone cannot be overstated.

Through Crosswords, I learned to realize that my point of view could be quite narrow and that asking for help was sometimes much more productive than trying to dig myself out of my own thinking.

My second major pitfall with this assignment was my unwillingness to ask for assistance in areas I did not have a good plan for. A single five minute conversation likely would have prevented hours of frustration and aggravation, staring at and a screen with no clue what to do.

Sometimes, if I spend too much time staring at a certain assignment, my productivity on that assignment begins decreasing because I lose sight of what I could be doing better, something that only external help can provide at that point.

The Crossword assignment taught me the value of admitting that I am stuck and going to another person, whether it be a teacher or a fellow student, for help and advice.

Something I learned about myself through this assignment was my lack of initiative to ask for help. I recognized the importance of asking the teacher for help in even when it seems late and that the help should've been asked a lot earlier. **This assignment demonstrated that I need to be more willing to ask questions even if it conflicts with my personality sometimes as there are no dumb questions or requests for help.**

I learned that I should ask for help before it's too late.

I realized the importance of asking others for help and collaborating to understand ideas and come up with solutions to some of the tasks.

I also learned about when to keep trying to work at something and when to ask for help. (…) If I asked for help when I was going in circles I would've been able to finish the lab much sooner.

I also reinforced the notion that asking for help is always a good idea. Thinking things through with another person is really beneficial in that other people may pick up on details that I may have missed.

Another thing that I learned about myself through crossword is that the best usage of class time is to ask questions and try to solve out algorithms with my peers rather than trying to just focus and code.

The fact that I pretty much asked no one and never shared my issues with anyone was easily the worst part. I learned how hard it is for me to ask for help and honestly I'm kind of thankful for it because I don't know how long this would have continued for.

I would say that this assignment helped me to recognize how much I rely on direction and instruction for programming assignments.

I have also learned that not everything comes easy to me and that I need to learn to open up to putting in more work with teachers and get help from other people. **It is important to ask for help if I am confused and stuck in a bad position because then my teacher will understand that I need help and will try to help me but he wouldn't know to help me if I do not ask him.**

As for honoring dissatisfaction, it depends on who or what I'm dealing with, but asking for help can be very hard/awkward for me.

I had some trouble admitting to myself that I needed assistance. I wouldn't say I have any sort of anxiety disorder, but things like that do make me very nervous and uncomfortable.

Something I learned about myself from this lab was the need to attend eighth periods if needed and ask for help.

I also learned how beneficial it is to ask someone for help immediately if I am stuck on one part.

I learned that getting help early in the time frame from teachers would make more sense than struggling alone.

I learned that I should get more help instead of trying to find my own way out of a difficult bug.

I learned that I have a severe aversion to asking for help. (…) **I think the reason for this is a false perception that I have that asking for help makes me seem less smart than my peers. This line of thinking is clearly false, as I ended up not finishing the lab, which is truly indicative of being less smart than my peers. The problems I faced could have been avoided if I had simply asked for help when I got stuck**.

I also found that brainstorming during school where I could talk with other people and planning out how I was going to code everything and then coding at home worked better for me than spending class time coding.

Time management:

If I couldn't figure out an error after 30 minutes, I would work on something else for 15 minutes, and this would usually help me figure out an error much faster.

I was not very good at managing my time on this assignment.

One thing that I learned about myself from this assignment is that in terms of coding I work best in really long chunks of time which sometimes makes it hard to work very effectively in class.

I learned that to manage my stress level for difficult assignments, I need to take more breaks to get my mind refreshed.

Breaking the assignment into parts and putting more time in at home periodically would have helped me do better in this assignment.

Through this assignment I learned how to better prioritize school assignments.

Something that I learned about myself is that working on the project in large chunks of time may not be advantageous to efficiently finishing the code. To accomplish future project I need to make sure that I am able to pace them out in a way that I am working on smaller sections of the code throughout the week instead of large sections in a designated period of time.

In the future, I must attempt to pace myself better even if I have a lot of other work that week.

I realized that I am not very good at managing and planning how to accomplish large scale tasks such as this.

In the time that I did this assignment I found a way that I could better manage my time. I started assigning specific tasks to time periods throughout the day.

I definitely learned the importance not procrastinating in this project.

I learned that my time management skills are absolutely not what they should be at this point in time.

I didn't really have any challenges with the labs in first semester or troubles with deadlines, so I think I just expected this lab to be the same difficulty.

From now on, whenever I get a new assignment I set miniature goals for myself before ever typing my code so that I know if I could allocate enough time for a lab.

If I had managed my overall workload from school better, I would not have waited until the last two days to finish my code, which would have made me less stressed.

For the future, after reading the assignment sheet, I'm going to actually designate days to work on the assignment that prevent last-minute procrastination.

In addition, during those days, it was also important for me not to do programming every waking second and to take breaks or do something else. I can keep up my focus pretty well for a good 3 to 4 hours, but if I get stuck on one thing for an hour or so, I find that going out for a walk or watching funny videos for 20 minutes is helpful in recharging brain power and decreasing frustration and stress.

I made sure that I had my time planned out all the way and made all of the intermediate deadlines (even though they were optional).

In order to do something like this, I feel that I worked best when I coded until I hit a wall, taking a break, and continued to work at a later time. A few times I can remember just walking through the halls to my next class, vaguely thinking about the project, and then having an idea, and quickly making a note of it on my phone to look at next time I worked on my project.

I realized through this project that I am no where near productive when I am working without sleep.

I think despite my difficulty with the assignment, I learned I was able to manage my time well and meet deadlines set by both Mr. Eckel and myself.

I'm going to start trying to set intermediate due dates for myself to work towards because I feel that I need to be able to motivate myself to work on assignments early on in the timeline, rather than let it all build up and stress me out.

Another method that worked for me was taking short breaks in between long hours of work.

When going through these stages where I was struggling with coding, I realized that the best use of my time would be to take my mind off programming and come back later instead of sitting and pondering the problem for hours.

I learned the importance of finding effective techniques of alloting my time for the lab. I don't necessarily mean time management, but rather making sure that I did not work on my code in one sitting for too long. Although I was eager to finish the lab, I realized that after 1.5 hours of work my progress would slowly decline to the point where it was hard to motivate myself to continue working on the lab. As a result, I found it easier to break my coding sessions into one in the morning and one in the afternoon, so that I would be constantly looking at my code with fresh eyes.

Something that I learned about myself is that it is always best to set a personal deadline before the actual deadline.

I just got really engrossed in trying to find my problem and coming up with a solution for it that I just did not think of taking breaks. In a way, I guess that was good since I was really focused on it, but eating is also pretty important sometimes. So, I need to learn how to be mindful while I am programming and not forget about other responsibilities and necessities.

One thing that I learned about myself was that I think I don't do well without checkpoints and guidelines.

I learned that I'm somewhat inflexible when it comes to problem-solving outside of what I'm accustomed too, and I have to take more time to consider different approaches, and not get pigeonholed into the uncreative tendencies that I've become used to.

I have learned to now look at code holistically and to make the necessary changes immediately rather than seeking a faster approach.

I have also learned that coming to a solution will not always take a specific amount of time that you have previously set. You can run into a number of issues that you have not accounted for.

This also ties in with my lack time management skills, since I'm sure if I had crafted a better and more specific plan on how to tackle my workload, I don't think it would be nearly as easy to get overwhelmed.

Through this assignment, I learned that time management and breaking the project up into smaller tasks is essential in order to finish the large assignment in an efficient manner.

Instead of writing code willy-nilly, I stopped after every line I wrote to take a quick second to realize how my approach could fail and what exactly I intended this line to do.

Taking 20-25 minutes of hard work and then abruptly taking a break in the middle of it simply detaches my train of thought and I constantly find it harder to get back into the proper attitude and recall all my exact strategies.

One thing that I need to work on is keeping a better schedule and seeing when assignments or tests or extracurriculars may add up and cause time management issues.

What I take away from this is that I need to better manage my time by breaking large projects into more manageable chunks that I have set deadlines for.

Additionally, something I discovered was that working on an assignment for a few hours on one day, making a note of where I left off and not touching it for a couple of days really benefited me on this assignment. During the "off days" I would find myself thinking a bit about how I would go about another part of the assignment or a strategy for improving something I had already written.

My experience from this assignment will tell me that it is far better to overestimate the time that would take to code than to underestimate, which will probably help me manage my schedule better.

Overall, I was able to learn that procrastination can really make an assignment much more difficult to complete, especially a coding assignment.

I learned that I need to be extremely focused in accomplishing one task, no matter how small, before moving on and slowly, the project will come together.

I found that by actually spreading out my work, I ended up a lot less stressed when it was actually time to turn in the lab.

I also recognized the need for me to be MUCH more efficient with the time given to me, both inside and out of class.

Motivation:

Through this lab, I learned that knowing exactly what was wrong kept my hopes up so that on my third go, I could correct the mistakes that I made in my first two tries.

Something interesting that I found out about myself is that I code much better when I'm in a really good mood and energized.

Something else that I learned about myself is that I need to have reasonable goals and expectations so that I can feel rewarded after working hard on something.

I learned that I don't think as clearly when I'm frustrated.

This assignment definitely taught me that I have a limit.

I need to learn how to take a step back and revisit the code at a later time, as that will allow me to face the problem with a clearer mind and new ideas.

I learned that I when it comes to big assignments, I tend to get overwhelmed.

I noticed that I finished (writing and debugging) codes much faster if I worked on them for less than an hour at a time, took a break, and then returned later.

I also noticed that, as I worked through the evening and the following day for hours straight, I would lose focus and become much, much less productive. Eventually, much of my time would be spent staring at the wall blankly, kind of thinking about the assignment and how to solve a certain problem, but not putting my full mental capacity towards it, as I had gotten tired out.

Something I learned about myself during this assignment is that I sometimes have a bad habit of getting too hung on up one issue, and instead of working on something else and coming back I just keep trying to fix the relatively unimportant problem.

Looking back on this project, two things about me are obvious. Firstly, I don't take enough time to plan, and secondly, I don't handle a lack of progress or stress very well.

I learned that taking your mind off one specific problem can have major benefits when you return to that problem.

Splitting the assignment into smaller tasks also made me feel less stressed by the whole assignment.

I learned that I work better under a bit of stress, either from the need for a good grade or from time pressure.

I learned that I can get a lot of work done when I am focused, but sometimes I need to take breaks in order to be able to keep thinking at the level I need to be thinking at.

I was too preoccupied with whether the implementation I had thought of would be efficient enough.

During my work time, I spent too much time worrying about the difficulty of the assignment, and not enough time trying to solve the assignment and having fun with a challenging problem.

Through this assignment, my perseverance and determination improved significantly.

This assignment brought to my attention how harmful intolerance of failure could be. Failure is normal and not being able to complete assignments to the quality I want will happen. Thus instead of viewing failure as something I need to avoid at all costs I should embrace it more and learn from it.

In the future, I intend to heed the lesson taught to me by this experience; instead of turning around when the path grows treacherous, I will forge ahead and push myself into the unknown.

I realized that stress brings out the worst side of me because the impending deadline and my coursework seemed to overlap and I never found the sufficient time to wholeheartedly work on this lab.

I learned that when I am under stress, I will work harder.

In my opinion, this lab was the first truly challenging lab of the year. It made me put more effort into the course than I did in any other lab this year.

Throughout this assignment I learned how much I hated getting rid of any old code, and how problematic this is. (…) When I finally convinced myself to let go of my junky old spaghetti code, it made life so much easier.

I came across many difficulties and was extremely worried and frustrated when my algorithm did not work as well as I wished or go as I planned. Many times, I had to go take a break, relax, and calm down before starting to work again.

I realized that I give up too quickly and I have to work harder on some assignments than others simply because I understand certain things better than I understand other concepts.

I would spend too much time trying to tweak a method that didn't work properly in hopes of improving it.

I think this assignment showed I need to get better at honoring my dissatisfaction, and also that the way I respond to being overwhelmed is super useless and bad.

I realized that I don't work very efficiently under stress.

I was pretty steadfast on getting my code to work the first way that I wanted to it that it took me too far into the project to realize that I should think of another strategy and ask for help in doing so.

One thing in particular I learned about myself was how it is very hard for me to begin these large projects and manage my time well, while it is generally much easier to work after I have started.

I also learned from this assignment that a common habit I have is putting off things that I dislike or get frustrated with, even if they are important.

I also learned that I prefer to handle stress by not thinking about the cause of stress.

I learned that I lose motivation just a bit too easily.

[This was] a reminder of the importance of emotional precedent on your work, and that you have to maintain a strong attitude to work effectively.

I also learned how hard it is to go back on an algorithm once it is already mostly coded. The feeling that what I coded did not work and would not work well enough ever, was pretty crushing.

I thought this assignment was a great way of showing what programming in real life would be like. I had to manage my time, learn how to organize my code, and debug efficiently.

A positive aspect was that I did not receive nearly as much stress as others were or than I expected.

I realized that I was not using my time efficiently because I would frequently get distracted by other work or I would easily get frustrated by my code not working.

When an assignment stresses me out, I tend to avoid it like the plague. I do everything that is not the assignment - other assignments, sleep, procrastinate...I leave the assignment to fester and pile up with other things until it's too late, and I end up desperately playing catch-up when the panic hits me.

My coding style/preferences:

In general, I learned that I shy away from viewing the problem as a whole, and instead enjoy considering it as broken up into a number of small problems.

I learn the best when I reach a "dead end" and work on trying to solve it.

I developed a nice bond with my code and even made a cheering routine thing that I do when my code is running the test cases (a little embarrassing but I would like to think it is helping even the slightest bit:)

I realize now that I like to have all of my "checkpoints" and "milestones" for an assignment laid out in front of me, being well understood for their difficulty.

I like to do one thing fully, then another thing fully, etc. instead of skipping between things.

Another thing I learned about myself is that I seem to avoid using a profiler for whatever reason, and that I really hate redoing things. Recoding everything seems like a whole lot of wasted work to me, so I try to fix the bad code as much as I can, but it wasn't good enough this time.

Rather than trying to write efficient code in the first try, I should write an algorithm that seems to work, for that's always better than having code that doesn't work.

After this project, I realized that I enjoy taking my time to code rather than trying to cram and rush the coding.

When coding, I also like to prioritize tasks that I know I can do quickly and correctly.

One thing I learned from this assignment is that often, the simplest solution is the best one.

The most important lesson I learned about programming from this assignment is that just knowing the basic solution is not always gonna work.

Due to the nature of the lab, it was essential for me, a visual learner, to be able to picture the board and what I wanted my algorithms to do before I coded anything. For example, drawing out various board states on paper and running through block placements myself helped me figure out a logical approach to placing blocks.

I realized that I tend to finish computer-science-related assignments quickly when I am working at home in a silent environment, as I need short periods of absolute silence to think of algorithms or optimizations.

During my work on the assignment I learned that I have a tendency towards focusing on issues in a depth method, as opposed to a breadth method.

Another essential skill that I was able to develop was version control. I accomplished this very simply using save as and labeling each version systematically.

I have started to keep a second version locally stored on a hard drive, as well as a version of my code stored on github so that this [losing the USB drive with his code] does not happen again.

I'm more powerful than I thought:

I have more resolve than I think and that I should underestimate my abilities less, which would possibly open me up to trying new ways of approaching problems in the future.

I also want to say that I learned how much persistence I had.

I also learned how to better deal with parent pressure.

One final surprising thing I learned about myself is how I am actually able to focus for a long time if I try.

One positive thing I learned about myself is that I handle stress pretty well.

I learned that I am relatively good at judging how long a particular task will take me.

I thoroughly enjoyed working through a difficult problem like this one and fixing my own problems because it made finishing the lab feel much more gratifying than other labs I've done.

Most importantly, I was able to prove to myself that I was able to attempt and solve a complex problem with what I had learned over the course of this year.

This assignment was one of the most interesting projects I have worked on in school ever, and I don't remember having so much fun working on any other project I have ever been assigned. (…) This lab also helped me learn how interested and willing I was to work in computer science.

I learned a lot about my ability to persevere and work through a rut from my experience with Crossword.

Crossword challenged me to the point that I was pushed outside of my comfort zone. Most classes have not had this affect on me and I think that crossword was a good opportunity to gain exposure to problems such as these.

This assignment taught me that I am more resilient than I had previously perceived. (…) By the time I had finished the assignment I was surprised how many obstacles I had successfully overcome.

What this showed me is that programs do not usually work the first, second, or even twentieth time. Rather it takes resilience to overcome the failures and continue coding.

This lab helped me to see how far I've progressed since the beginning of the year. I was happy to see that I was able to come up with a lot of creative ways to code different parts of the lab and to debug the errors I was getting.