CODENAME: JIGGLYPUFF

Software Plan

Skylar Fritzky
Mitchell Shaw
Jorge Koifman

Appendix

-Complete Class Diagram

Software Plan

- **Overview.**
  - **Project Summary.**
    - **Purpose, Scope, and Objectives.** The objective of this project is to develop a software product that will assist rising students in the Systems track in the Computer Science Department at Columbus State University. This software is aimed at visualizing complex algorithms taught in the Algorithm Analysis and Design course. The product will have a user friendly interface that will allow the professor to click on the algorithm he would like to demonstrate, and allow custom input to show a real time visualization of the problem at hand. By implementing a step-by-step functionality and showing the movements of the trees and graphs, it will allow the students to better understand how the algorithms work without having to draw them out by hand for each individual problem. This software is aimed at the professor to be used as a teaching aid for the students. Algorithms that will be used for visualization include: **Heap Construction**, **Horspool's Algorithm**, **Prim's Algorithm**, and **Huffman's Algorithm**.

    - **Assumptions and Constraints. Constraints include the following:**

      - The deadline must be met.

      - The product must be reliable.

      - The architecture must be open so that additional functionality may be added later.

      - The product must be user-friendly, and easy to follow as the graphs and trees move throughout the algorithms.

      - The product must be available only to professors, as students could potentially abuse it.

- Project Deliverables. The complete product, including a Readme.txt file that includes how to use the software, as well as contact information of each team member in case a problem arises for the user, in this case the professors, will also be given.

- **Schedule Summary.** The duration of each phase, the requirements for the team, and the formal documentation deadlines are as follows. The core team will consist of three software engineer students, Skylar, Jorge, and Mitchell. With a smaller team, all team members will be working on the software together. Skylar, Jorge, and Mitchell all will be working on the software free of charge as an educational experience to further grow our software development skills. A working week for the team consists of 168 hours, during this time we will dedicate 20 hours of work a piece on our software project. The schedule is as follows, and will be strictly maintained during the lifecycle and development of this product.

    - Initial Software Plan/Brainstorming (2 weeks, all team members)
        - Dates: Jan 9 - Jan 24
        - Mitchell's Tasks: Coordinate emails and create layout for software plan document. Begin research into how we will be implementing the algorithm design into our system and how the input will be used to demonstrate each. Communicate with other team members on their portions of the design respectively and lay general guidelines of what we want to do.
        - Jorge's Tasks: Begin research of GUI implementation and design layouts of software, communicate with Skylar and Mitchell about possible layouts and designs. Work on scheduling and metrics for Software Plan. Learn how to use Visual Studio in an effective way to allow a rapid prototype phase for the software. Brainstorm layout with team, showing sketches and general functionality.
        - Skylar's Task: Begin brainstorming animation details in Unity and finding appropriate Sprites for animation work. Communicate with team members about how the design will need to move and track the progressions of each step. Define the theme of the animations, including color and size.
        - Group Tasks: Collaborate on design and figuring out what needs to be done first to set the framework up for the application. Decide what will be doing first and in what order we will need to implement each person's individual contribution.

    - Milestones: Solidifying our plan of attack is the only milestone for this project, and deciding how we will approach the application. Once completed, we will revisit this plan to perform any necessary changes.

- Requirements workflow (2 weeks, all team members)
    - Dates: Jan 24 - Feb 7
    - Mitchell's Tasks: Creating the business model for the plan as well as communicating with team members about who will be doing what part of the report. Mitchell will be responsible for creating the initial use case diagrams for the business model as well as the business model summary. He will also work with the other team members for a step by step description of the use cases.
    - Jorge's Tasks: Creating the glossary for the project that is required for the document. Jorge will work closely with Mitchell and Skylar to create the necessary use cases for the GUI implementation.
    - Skylar's Tasks: Communicating with the team that is unfamiliar with manipulating graphics about use cases he will need and be using to produce the animations. Skylar will work closely with Jorge and Mitchell to figure out how this portion of the project will impact the requirements and how they relate to each function in the use case. Skylar will also be responsible for the reports section of the requirements plan.
    - Group Tasks: Collaboratively deciding what use cases will be needed and the initial domain of the project. Understanding how each part of the application and how they relate to each other in the scope of the project will be planned in this section and the appropriate use cases for each section.
    - Milestones: Upon completion of this stage of the plan we will have decided our major design components needed to implement into our final application. By doing so, the rough outline of what we will need to implement will be decided and work can begin on the next phase, analysis and realizing the use cases.

- Analysis workflow (2 weeks, all team members)
    - Dates: Feb 7 - Feb 21
    - Mitchell's Tasks: Mitchell will work closely with Jorge and Skylar to decide what classes will be extracted into the

final application with each individual component that the team member is doing. Working with the team members, he will be able to define the boundary, entity, and control classes to begin modeling the use cases.
    - Jorge's Tasks: Jorge will work closely with Mitchell and Skylar to decide the classes needed to make the GUI function with the rest of the application. Jorge will be responsible for informing the group of how he plans on laying out the GUI and how his classes will function with the rest of the team's classes.
    - Skylar's Tasks: Skylar will work closely with Mitchell and Jorge about the classes needed to implement the animation sequences into the application and how these classes will relate to the rest of the application.
    - Group Tasks: Identifying and realizing the uses cases will be implemented by the entire team as well the documentation for the classes individually and the complete class diagram. Scenarios and CRC cards will be worked on as a group due to the scale of the project.
    - Milestones: Completion of the Analysis stage represents the end of the planning phase in regards to classes, this will mark the time where we can begin to think of relationships among the classes and how we overall implement these classes into a final product.

- Revised Software Plan (2 weeks, all team members)
    - Dates: Feb 21 - Mar 7
    - Group Tasks: Revisiting the software plan will be done by the entire group to decide the last workflows and make any revisions needed to have complete documentation of the process we will be following. In this stage, any documentation that is changed will be formally addressed and marked in a document, keeping all previous revised versions.

- Design workflow (3 weeks, all team members)
    - Dates: Mar 7 - Mar 28
    - Mitchell's Tasks: Mitchell will be the spearhead on the majority of this portion of the project. He will be in charge of coordinating the design of all the classes so that they are cohesive and work together. How the algorithm's classes interact with the GUI and the animation aspect of this in relation to the algorithms and how they will work.

The pseudocode and layout of the algorithm implementation will fall on him.
    - Jorge's Task: Jorge is still on the forefront of metrics and will be in charge in of writing the pseudocode for the GUI and it's pertaining classes. Jorge will also be keeping track of the hours we are working on this document so that we can further analyze the effort we are putting into the project.
    - Skylar's Task: Skylar will be responsible for generating the pseudocode for the animations that will be happening, and the stepping method for each corresponding algorithm. Working closely with both Mitchell and Jorge. Jorge will need the assistance of Skylar to utilize the buttons in the correct way and with Mitchell to make sure that the stepping of each algorithm is properly implemented.
    - Group Tasks: The group will be responsible for editing the pseudocode for the classes in an efficient, clean way, and checking the work of others. In this workflow, we will check each other's pseudocode collaboratively in an effort to make this application highly cohesive. How we design the document is extremely critical for the next step, so taking our time and checking each other here is critical.
    - Milestones and Goals: Milestones here include generating the pseudocode for every class, in doing so we will be able to have a clear cut path for our implementation workflow. Another milestone for this goal we will be reaching is the official halfway point of the project, we will be over halfway through development of our first real application.

- Implementation workflow (3 weeks, all team members)
    - Dates: PENDING
    - Mitchell's Task: Mitchell will be in charge of implementing the backend logic for the algorithm's design. He will use the design document to implement the logic behind how the algorithms will work, and how they will coincide with Skylar's animations and the GUI Jorge is developing.
    - Jorge's Task: Jorge will be in charge of implementing the GUI and the functions of the GUI using the design document as a template. Creating the GUI is a central part of the project, as this is what the user will be in contact with. Jorge will be responsible for working closely with Skylar and Mitchell to insure that the algorithms will run

based on how the user interacts with the menus and sub-menus.
        - Skylar's Task: Skylar will be in charge implementing our animation sequences. This includes all changes that will be made to the visual representation of each algorithm and adding in the capability to backstep at any point during the visualization. Skylar will work closely with Mitchell to make sure that the implementation of this coincides with the movement logic that he will be implementing.
        - Milestones: This phase will be the completion of creating the application physically, making a huge milestone for the group. This will be the first application that has been created by all three students using the Unified Process. The ending of this phase will represent near completion of the project as we prepare for our final demonstration to our colleagues.

    - The total development time is 16 weeks

    - No fee will be charged during the lifecycle of this software, it is strictly aimed to be an educational experience for the team and to better the learning community that we are involved in at Columbus State University.

    - Overseeing the production in each phase will be our professor, Dr. Yenduri, who will be the lead on determining the quality of the software provided and as an aid in the life cycle development. He will work one day out of the week for an allotted time to answer any questions or rising concerns that the team may have.

- **Evolution of the Project Management Plan.** All changes to the software project management plan must be agreed upon by all team members before they are implemented. Strict documentation of all plans will be put into place. A timestamp will be required for all changes as well as an artifact kept documenting the change in the plan. Any changes that are to be made to the software project management plan are subject to approval by Codename: Jigglypuff and Dr. Yenduri. We will revise this software plan accordingly as the development of the software progresses.

- **Reference Materials**. All documentation will conform strictly to the Unified Process that is being utilized, as well as a Read-Me.txt file that will be created for the user to understand the purpose of the software. We will utilize the API for Java as well as the creation of our graphs and trees using Visual Studio. All testing standards will be held at

the highest level possible as to insure that the software can be reliably used in an academic atmosphere.

- **Definitions and Acronyms.** Codename: Jigglypuff - Our organization; Dr. Yenduri - Our professor

- **Project Organization.**
    - **External Interfaces.** Skylar Fritzky, Jorge Koifman, and Mitchell Shaw will perform the work on the software. During development Jorge Koifman

    - **Internal Structure.** The development team consists of Skylar Fritzky (Games track), Jorge Koifman (Games track), and Mitchell Shaw (Systems track), and Dr. Yenduri (Software Engineer professor).

    - **Roles and Responsibilities.** Codename: Jigglypuff as a team is represented by two Games majors and one Systems major. We want to play to the strengths of each team member in their own way. Skylar, the first Games major, will be responsible for implementing and creating the movements and animations of the trees and graphs. Skylar will be implementing custom Sprites and animations to do so. Jorge will be responsible for creating the graphical user interface and linking the features of the GUI to Skylar's Sprites and animations. Mitchell will be responsible for linking the GUI buttons to the corresponding logical algorithms and allowing Skylar's animations to be presented to the user. Jorge and Mitchell will both use Visual Studio for the GUI creation and the logical backend of algorithms.

- **Managerial Process Plans.**
    - **Start-up Plan.**
        - **Estimation Plan**. Stated prior in Section 1.1.4 the development allotted for this project is 16 weeks of work. The total development time is based on Dr. Yenduri's discretion, the window for our team Codename: Jigglypuff to complete this project. The total internal time is based upon an average computer science student's schedule and the timeline created by Dr. Yenduri. For our estimation of nominal effort in the project will use Intermediate Cocomo to distinguish the effort needed to create the application. We will assume that this application is organic in nature, being small and straightforward, we will work under these following assumption in calculating the effort for the team. The reliability of the software will need to be high, the product complexity is nominal, execution time constraint is high as we have been given strict deadlines, we will assume the personal attributes of the team is considered nominal at our current level in our degree field, as well as nominal for project attributes. In our analysis phase we will use this to estimate our nominal

effort for the project, we will work under the assumption that we will include 2,000 source statements and will edit this number provided the complexity of the project increases or decreases.

- **Staffing Plan.** Codename: Jigglypuff is a three member team, with an additional external professor, Dr. Yenduri. Each member of three man team will work collaboratively through each phase of the workflow to insure speedy development and sound logic. Dr. Yenduri will be actively checking each workflow for a total of one class day per week to insure that the quality is up to academic standards. Skylar, Jorge, and Mitchell will all be working as lead designers and programmers in each workflow. For the final phase, Skylar, Jorge, and Mitchell will all work as testers.
- **Resource Acquisition Plan.** All hardware, software, and CASE tools for the project have been made readily available by our sponsors at Columbus State University. The product will be delivered to the professor of the Algorithms class here at Columbus State University.

- **Work Plan.** The work plan has been formally documented in the schedule portion of this document, please revisit this for a broken down, formal declaration of how each team member will be working in regards to the schedule and work plan. This outline is a generalized version for our records.
  - **Work Activities and Schedule Allocation.**
    - Initial Software Plan/Brainstorming (2 weeks, all team members)
      - Creating initial idea of working software and functionality that would be implemented.
      - Rough sketching of the GUI and tree interface
      - Create formal documentation of Initial Software Plan

    - Requirements workflow (2 weeks, all team members)
      - Create and develop all of the requirements needed to make the software functional
      - Start brainstorming the relationships among the software to user to nail out the details of what we will be needed to be implemented to meet the needs of the professor and the students learning.

    - Analysis workflow (2 weeks, all team members)
      - Begin building the foundation of the software and identifying use case for the software.

    - Design workflow (2 weeks, all team members)
      - Structural design of how the classes will be set up and the relationship and interactivity among them.

        - Revised Software Plan (2 weeks, all team members)
            - Revisit software plan to determine that the Unified Process is being upheld and that we have no major changes to make to our plan.

        - Implementation workflow (3 weeks, all team members)
            - Creating the application itself based on the documents that we have created for design. This workflow will be the most complicated, but will also have the most supporting documentation.

        - Testing workflow (3 weeks,all team members)
            - This workflow is to get rid of any kinks or errors that have risen out of implementation workflow. We will be testing the software application for any inconsistencies and perform our unit tests.

        - The total development time is 16 weeks,

- **Resource Allocation**. Skylar, Jorge, and Mitchell will work separately on their assigned artifacts. Skylar will be using Unity to create graphical representations of the trees and graphs using a series of Sprites. Jorge will be responsible for creating the graphical user interface. Mitchell will be in charge of creating the algorithms and integrating them with Jorge's GUI and Skylar's Sprite Creation to allow the program to function. The majority of the time will be focused on integrating each team member's individual components with each other. Thorough testing of outlier cases will be used to insure that no set of inputs will cause an unexpected error.

- **Control Plan.** All the major changes that affects the goals are to be approved by the team and documented. Instead of having outside quality assurance personnel involved, each person will test another person's work products. The team will be responsible for assuring that the project will be completed in time and all goals have been met. We will accomplish this by having daily communications in place with all the developers to insure that the timeline is being met. On each meeting each member will show their progress and discuss any problems they have encountered. The team will then determine if their progress is going as expected and if they are following the specification document and the project management plan. All major problems encountered by the team will be recorded in a document to insure that these problems are addressed.

- **Risk Management Plan.** To mitigate and track all risk factors a few steps must be taken: When developing the software, the following constraints will be put on the developer's: Lack of assumption that the user has the skills necessary to use the software, provide a user interface that is easy to follow, do not over complicate user experience, ensure that all possible test cases have been tested. It is the programmers responsibility to ensure that he/she checks all code and verifies that the it is fully tested before passing the software to the professors. Ensure that when developing the software we produce a product that will integrate smoothly with the learning environment that has been created in the Algorithm Analysis and Design classroom. It is also imperative to keep this software out of the hands of the students of the Algorithms class as to not be used a solution tool as opposed to an educational visualization tool. It is the team's responsibility to ensure that any all and issues are worked out before the product is finalized by the Algorithms professor and Software Engineering professor. Lastly, because we want this software to have the ability to grow, we want to develope it as lightly as possible. This means that all classes must be created in an efficient manner.

- **Technical Process Plans.**
  - **Process Model.** We will be utilizing the unified process model and associated techniques.

  - **Methods, Tools, and Techniques.** We will be writing this software using the unified process and coding this software in Java. All software will be written using the Visual Studio environment. Visual representation of the trees and graphs will also be created utilizing Visual Studio and Unity Sprites.

  - **Infrastructure Plan.** The product will be developed on Windows based machines using the latest version of Visual Studio. Lab computers will be utilized as well as all existing infrastructure in our personal homes. Once the product is complete we will give a copy to the Algorithms professor at Columbus State University as well as our Software Engineering Professor.

  - **Product Acceptance Plan.** We will ensure that university and our professor will accept the software via the Unified Process.

  - **Problem Resolution Plan.** In the encounter that a large scaled problem is found within one of our artifacts, we will keep on file all of the documentation of the software plan, including any revisions during the multiple stages of the Unified Process. In the occurrence of a found

discrepancy, the documentation will be noted to include the change in all formats, and if necessary, a rollback of any versions of the current phase of programming will be made. All versions of the application will be saved in their original condition, in case of a rollback occurrence.

- **Supporting Process Plan**
    - **Testing Plan.** The Unified Process will be utilized to implement the testing plan. We will focus heavily on making sure that a sample input is incapable of breaking the logic of our software, as this software focuses on the visual, logical proof of an algorithm being solved. Execution based testing will consist of outlier testing and performance in a time situation, given the spacetime of a particular algorithm. Non-execution based testing will include making sure that our graphical interface does not lead to crashing in the application itself, and that animations do not create crashing of the application, , this is considered destructive and stability testing. We want to try and keep the software size in terms of storage to a minimum so that it is lightweight and portable. We will be performing manual tests on our application, as opposed to automated testing. Mitchell and Skylar will be responsible for testing out the functionality of the algorithm visualization by using a set of standard outlier data that we will deem later in the revision of this software plan. This data set will be created with the means of creating a fault in the application, in an attempt to break the algorithms design and force a fault in the application. The metric we will be using in this case will be a cumulative total of the faults found in each recurring version. By tracking this, we will aim to lower the faults of each version that is iterated until the fault list reaches its smallest value, informing us that we are closer to a better application. Jorge will do something very similar in his approach to testing, instead testing if the GUI responds correctly to different types of user controlled input, rapid clicks, double clicks, and negative space clicks, this type of testing that will occur is considered usability testing . Tracking any faults in this in the same document as Skylar and Mitchell, Jorge will be able to distinguish if the faults are getting lower with each iteration of the application.

    - **Documentation Plan.** Any and all documentation will abide by the Unified Process. Documentation will be done by all members for their separate tasks. Mitchell will complete documentation in regards to what the different algorithms do, how they work, and other simple instructions needed to understand the algorithms. Jorge and Skylar will complete the documentation for the code behind the GUI as well as additional walkthrough instruction for how the GUI works, moves and the steps needed to understand the application. Skylar will also document the steps taken to create the graphical representations of the trees to ensure that

the process can be recreated in the future. Documentation will be done while doing the assigned tasks, but will be reviewed every 2 weeks to ensure quality. All documentation will be stored in a folder with the metrics. The group will be responsible for logging the time that they work on every workflow, as well as include at the bottom of each completed document a revision log. This revision log will be very important as we will be able to keep track of the ripple of any changes that may be caused by making a slight adjustment to the overall application.

- **Quality Assurance Plan and Reviews and Audits Plan.** All code will be individually tested by the assigned developer. Once the developer has completed the code it will be sent to Dr. Yenduri whom will once again test the code.

- **Problem Resolution Plan.** Due to our development team being small, all problems must be relayed to all developers. Recommendations will be passed between all developers and a resolution will be determined. Once the developers have agreed upon the resolution it will then be implemented.

- **Process Improvement Plan.**

- **Metrics**
  - **Requirements Workflow** (2 weeks, all team members)
    - For the requirements workflow we will be keeping track of volatility and efficiency metrics every day until this workflow is complete.
      - **Volatility**: We will be keeping a record of how frequently the requirements change in order to determine the rate on which the team converges on requirements. By keeping this tally of changes that are made, we are able to see the evolution of the application, allowing us to further our design to a more refined state. However, too many changes, a high number for this metric, may represent to the team a loss of scope on the needs of the application. Jorge will be in charge of documenting each and every change made to the requirements document.
        - Based on the revision history of this document, there were a total of 141 changes made during this workflow.

In between all the team members, 74 changes made by Mitchell, 11 changes were made by Skylar, and 62 changes were made by Jorge. With a total of 141 changes throughout the whole workflow.
More specifically, the number of changes made on each day are as listed below:

01/31/17:	11	Changes
02/01/17:	2	Changes
02/03/17:	2	Changes
02/05/17:	90	Changes
02/06/17:	35	Changes
02/07/17:	1	Changes
Total:	141	Changes

    - **Efficiency**: We will be keeping track of which part of the effort is being spent on reworking said requirements in order to be able to measure our efficiency. By analyzing the efficiency of the team using Intermediate Cocomo, we are able to realize the effort we are spending on the project as a whole. By seeing the hours put into this particular part of the project, we will refine the scope even further of the application and understand at a higher level exactly what is required of the application. Skylar will be in charge of documenting these changes and man hours worked utilizing the session counter in the Google Doc.

- **Analysis Workflow** (2 weeks, allthis team members)
    - For the analysis workflow we will be keeping track of efficiency metric every day until the end of this workflow.
        - **Efficiency:** We will be keeping track of which part of the effort is being spent on reworking the diagrams of the Analysis workflow in order to be able to measure our efficiency during this workflow. By analyzing the efficiency in this stage the team will realize the amount of effort put into analyzing exactly how the application will work together. By measuring the effort put into this workflow, we should see the evolution of the diagrams we will be

creating for the classes and use cases, this will allow the team to understand the importance of this particular workflow. A high efficiency rating here is critical, as this portion of the Unified Process is one of the more robust sections, meaning the team will need to work effectively together to strive for this metric to be at its highest point. Mitchell will be in charge of measuring this metric, and will keep a log of the hours worked on reworking any part of this workflow, as well as the hours spent working on the initial workflow.

- **Design Workflow** (2 weeks, all team members)
    - For the design workflow we will be keeping track of the the cohesion between our classes (LOCM4). We will be measuring this every day a new class is made until the end of the workflow.
        - **LOCM4 (Lack of Cohesion Methods):** We will we be keeping track of how cohesive our classes are by measuring the number of responsibilities each of our classes possess. By utilizing this metric we will be able to realize which classes need to be refactored or split in order to maximize the cohesion of our classes and achieve greater performance overall. Jorge will be in charge of gathering this information and taking notes as to which classes should be refactored or split so that each class obeys the “Single Responsibility Principle.”

- **Implementation Workflow** (3 weeks, all team members)
    - For this workflow we will be keeping track of the Size (KLOC) metric every day until the end of this workflow.
        - **Size:** We will be keeping track of the LOC the program contains during each of the steps of the implementation workflow. With the data gathered he will be computing the Errors per KLOC, and Defects per KLOC. By evaluating these metrics we will end up with the Error rate, and the Defect rate that may point us towards a better quality process. This metric is extremely important to our efficiency rate that we will be using, Intermediate Cocomo, we are currently working under the assumption that the

project will take 2,000 lines to create. By having an extremely accurate LOC count, we can more effectively calculate the effort of the team. The group will work together on this portion of the metrics and will convene to make sure that this metric is accurate, therefore allowing our nominal effort rate to also be accurate.

- **Testing Workflow** (3 weeks, all team members)
  - For this workflow we will be keeping track of the Efficiency metric every day during until this workflow is completed.
    - **Efficiency**: we will be keeping track of which part of the effort is being spent on reworking said requirements in order to be able to measure our efficiency. We will then utilize the data collected to calculate the Rework effort ratio (Actual reworks efforts spent in the phase / total actual efforts spent in that phase) X100. Mitchell will be in charge of this metric, and we will realize that if the number is larger than the average efficiency for this phase was very low.

- **Additional Plans .** Additional components:
  - Security. Only the professor of the Algorithm Analysis and Design class will be given a copy of the

  - Training. The algorithms professor will be given a demonstration on how to effectively demonstrate the different algorithms to the class with the software we are creating. We will demonstrate thoroughly each algorithm that is including in the application and how to use them in a learning environment. An in-depth session will be given to the professor to the show benefits of using our application as well as a formal walkthrough of the process we implemented into the design. We will attempt to define any outlier cases that may potentially cause an error in the application. The Read-Me.txt that will be included with the application itself will outline the majority of our session and will be user friendly and in depth to a trained Computer Science professor. Showing how to navigate the UI as well as how to input data into the application for

display will be a heavily visualized concept in the demonstration to the professor.

- Maintenance. Corrective maintenance will be performed by the team during the lifecycle of our time at Columbus State University, and if we as a team decide to continue development of the project post-graduation.

Appended Material

Estimation:

For our estimation plan we decideed to use Intermediate Cocomo. Our total allotted time for this project is 16 weeks. Using Intermediate Cocomo, we have to state our assumptions to calculate it accurately.

Cocomo Assumptions:

- Organic Application
- Reliability = High
- Product complexity = Nominal
- Execution Time = Very High
- Personal Attributes = Nominal
- Project Attributes = Nominal
- Estimated Source Statements= 2,000

Cocomo Calculations

Overall Estimation: $3.2 \times (2\ KDSI)^{1.05}$ = person months

Step 1: 3.2 * (2.0705) = person months

Step 2: 3.2 * (2.0705) = 6.63 person months

Step 4: Modifiers: 1.15 *1.3 = 1.495

1.495 * 6.63 = 9.912 person months

Execution Based Testing:

For our testing we will be using two types White-box testing methods. Statement coverage will be the first, which is basically a means to include extreme outliers and sanitize them as they are included from user input because it involves the execution of all the statements at least once in the source code.. Branch coverage will be the latter, which is simply testing statements to determine if all lines are executed upon given input, due to the fact that branch coverage aims to ensure that each one of the possible branches from each decision point is executed at least once and thereby ensuring that all reachable code is executed including every true and false decisions. The testing pattern includes Jorge testing Skylar's code, Skylar testing Mitchell's code, and Mitchell will test both of his members' code.

Non-execution based testing:

Our team will utilize the inspection process for our non-execution based testing. As opposed to walkthroughs, this will allow us to track our metric of size, as well as implement a metric to determine the fault density. Skylar will act as moderator, Jorge as the recorder, and Mitchell as the reader. The moderator is the leader of the inspection, and is in charge of planning the inspection and coordinating it. The recorder is in charge of documenting all defects that are found during the inspection. The reader reads through the documents, one item at a time, cross examining all the defects the other inspectors have pointed out.

Using the inspection process, we discovered several defects that were causing some errors and buttons to not work properly. Upon discovery, these defects were taken care of and fixed to increase efficiency.

CODENAME: JIGGLYPUFF

Requirements Document

Skylar Fritzky
Mitchell Shaw
Jorge Koifman

Initial Understanding of the Domain

CODENAME: JIGGLYPUFF is an application primarily for professors over the Algorithm Analysis and Design course. In this course students are asked to show each step in a binary tree traversal for a number of algorithms, but to accomplish this task, it

would take quite a bit of time as well as a number of sheets of paper. Additionally, the resources for help do not show the individual steps, but instead show the start and the end. The application's goal is for professors to be able to show each individual step by selecting one of the provided algorithms, inserting a single number, and then selecting next step to show the next step in the process of creating or traversing the tree, this process will recur until the input data is over. This resource is aimed at providing students a visual aid in learning complex tree traversals and upper level algorithms.

CODENAME: JIGGLYPUFF Glossary

Algorithm - a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.

Jigglypuff - codename for the application that is being built.

Heap Construction - an algorithm consisting of heap construction, this algorithm will build a heap structure either from the top down or bottom up.

Horspool's Algorithm - an algorithm for finding substrings in strings.

Huffman's Algorithm - A data compression technique which varies the length of the encoded symbol in proportion to its information content, that is the more often a symbol or token is used, the shorter the binary string used to represent it in the compressed stream.

Prim's Algorithm - a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

Business Model

Business Model Description:

At the beginning of a class involving complex traversals of algorithms, the algorithms professor will turn on his computer and navigate to the CODENAME:Jigglypuff application. The professor will double click the application and it will begin to run. The professor will then choose what algorithm the class will be covering by clicking it's appropriate tile on the screen. A submenu will

pop up for each of the corresponding tiles that are placed on the screen, reading **Heap Construction**, **Horspool's Algorithm**, **Prim's Algorithm**, and **Huffman's Algorithm.** On each of the submenus upon clicking a tile, the professor will be presented with two text boxes and a larger square in the middle. The left hand box will be the text input, asking for a number as input. The right will be a list of all input that has been made. The professor will press the enter button and the application will add this to the currently selected algorithm. In the middle of the screen on input, the application will add this to the tree. This will continue until the professor is out of input, and each time will show the new input. In the case of a wrong input or incorrect number, a step back button will be placed on the screen. This step back button will revert the tree to the previous entry and delete the last input. To return to the main screen, a main menu button will be added. On clicking the main menu button, the professor will be directed back to the start of the application. This scenario holds true for all sub menus that will be presented. Certain sub menus will have different types of sub boxes that will be presented before input selection is made possible. The professor will choose the corresponding boxes that meet the criteria for the algorithm he/she will be demonstrating.

The first use case: ***Application Opening****,* we have one actor, the professor, which is the professor interacting with the application to open it.

The second use case: ***Algorithm Tile Selection****,* we have one actor, the professor, interacting with the main menu of the application to select an algorithm to start demonstrating.

The third use case: ***Algorithm Input****,* we have one actor, the professor, interacting with the submenu of the corresponding tile that has been chosen for demonstration. The professor will input a single numerical input that will be saved into the corresponding list and displayed on the screen following the format the algorithm specifies.

The fourth use case: ***Algorithm Input Entry****,* we have one actor, the professor, interacting with the submenu of the corresponding tile that has been chosen for demonstration. The professor will press the enter menu tile that will allow him to add the input to the list and to be displayed under the conditions of the algorithm.

The fifth use case: ***Algorithm Backtrack****,* we have one actor, the professor, interacting with the submenu of the corresponding tile that has been chosen for demonstration. The professor, on an incorrect input that has been entered into the algorithm, will have the ability to remove the last numeric input and remove it from the tree and the list.

The sixth use case: ***Main Menu Selection****,* we have one actor, the professor, interacting with the submenu of the corresponding tile that has been chosen for demonstration. The professor will have the option to click a main menu button at any time to revert out of the submenu that he is in and go back to the main menu screen and select a new algorithm.

The seventh use case: ***Submenu Options****,* we have one actor, the professor, interacting with an appropriate submenu such as the **Heap Construction** algorithm. The professor will be given a series of tick box options to select how the algorithm will be implemented, such as top down or bottom up. He will then be sent to the corresponding submenu and the algorithm will follow the guidelines of these tick boxes.

Business Model Case:

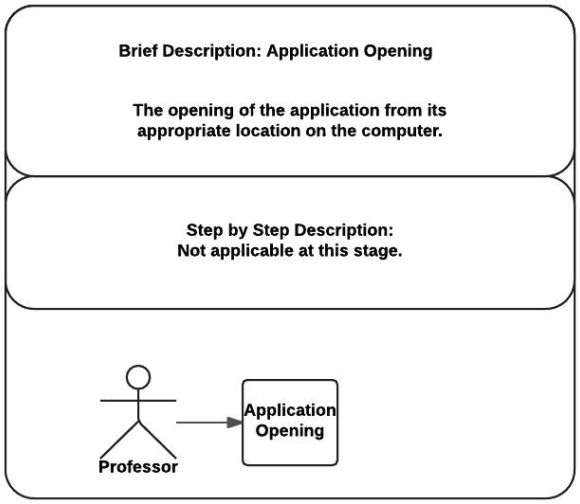


Figure 1

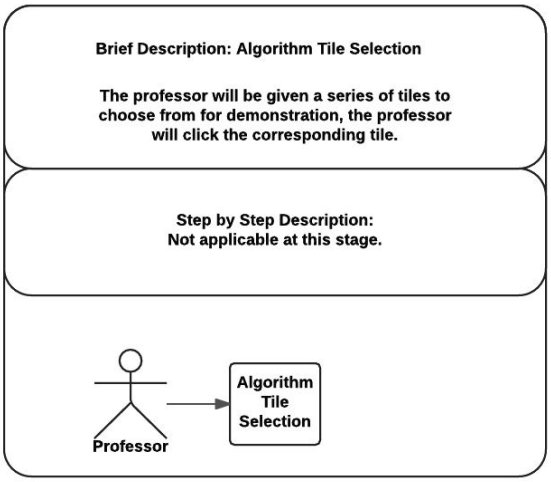


Figure 2

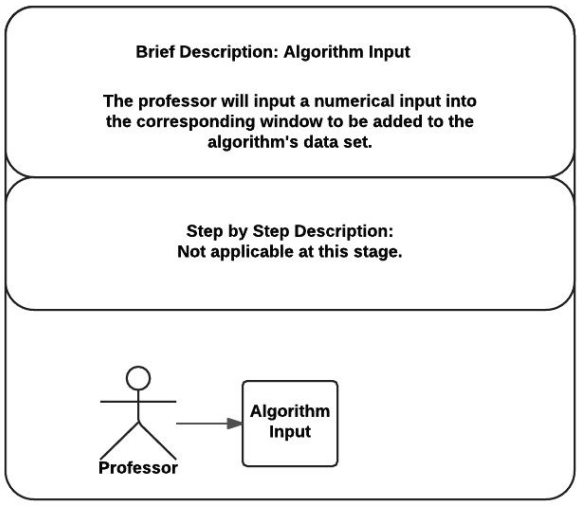


Figure 3

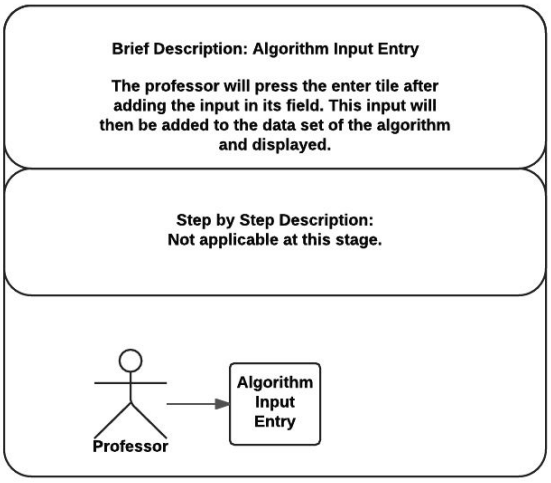


Figure 4

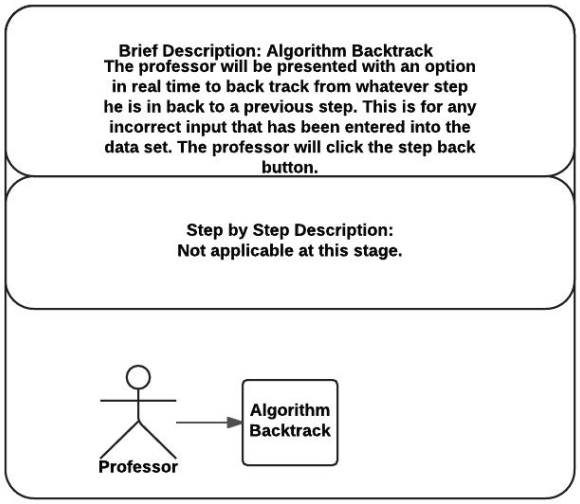


Figure 5

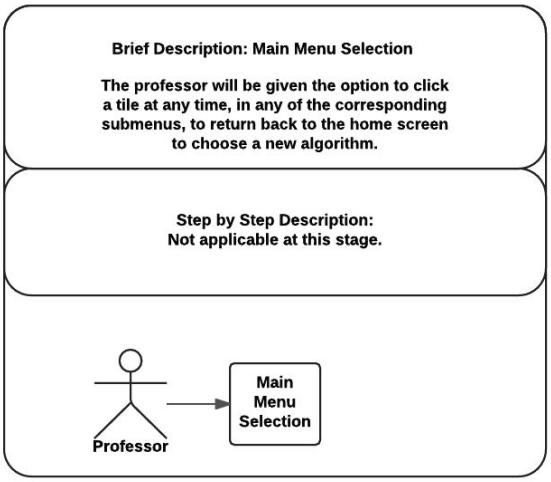


Figure 6

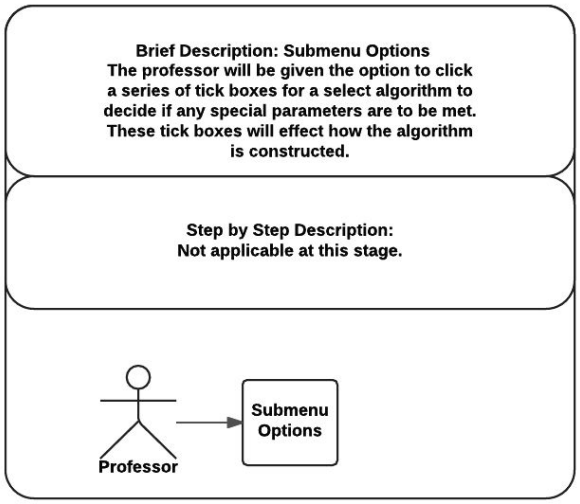
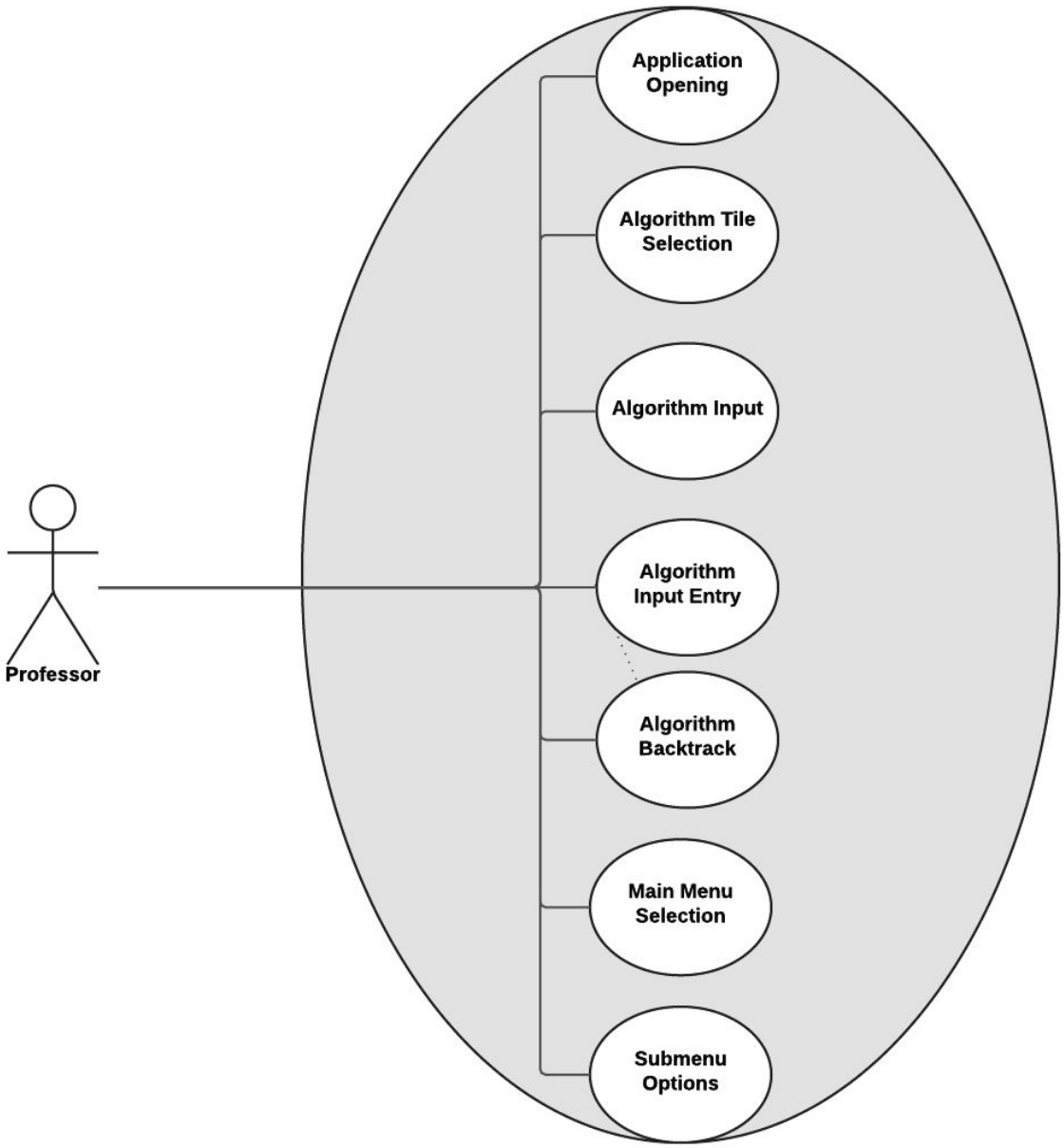


Figure 7

Initial Requirements

➢ Cases



➢ Step by Step Description of Business Use Cases

- **Figure 1:**
  i. The professor finds the application in his file system.
  ii. The professor double clicks the .exe file and runs the program.
- **Figure 2:**
  i. The professor chooses the algorithm for demonstration.
  ii. The professor clicks the corresponding tile to the algorithm that he has chosen.
- **Figure 3:**
  i. The professor is now in the submenu of the application corresponding to the tile that has been chosen.
  ii. The professor will see two text blocks, one for input, one for the data set.
  iii. The professor will enter the input value that he would like to start with into the field that says “Input”.
- **Figure 4:**
  i. The professor, after inputting a numerical, will see the enter button.
  ii. The professor will press the enter button.
  iii. Upon pressing the enter button, the input will be added to the algorithms structure and shown in the data set text box as well as in the visualization square.
- **Figure 5:**
  i. The professor will have the option to backtrack on a input mismatch.
  ii. The professor will see the tile that says “Step Back”.
  iii. The professor will click the “Step Back” tile, deleting the last input he made from the data set text box and from the visualization square.
- **Figure 6:**
  i. The professor will have the option to return to the main menu screen at any time.
  ii. The professor will see the tile that says “Main Menu”.
  iii. The professor will click the “Main Menu” tile, sending him back to the main algorithm selection screen to start over, or select a new algorithm.
- **Figure 7:**
  i. The professor will be presented with a window that has tick boxes for algorithm special parameters.

ii. The professor will select the tick boxes that apply to the algorithm, if any.
iii. The professor will be sent to the corresponding submenu for algorithm visualization.

Revised Requirements

➢ Cases

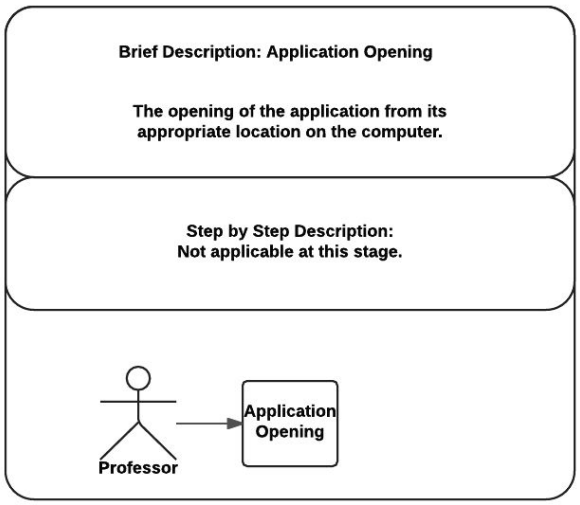


Figure 1

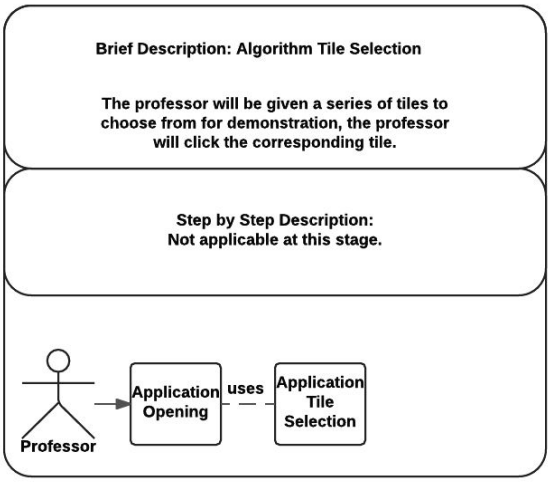


Figure 2

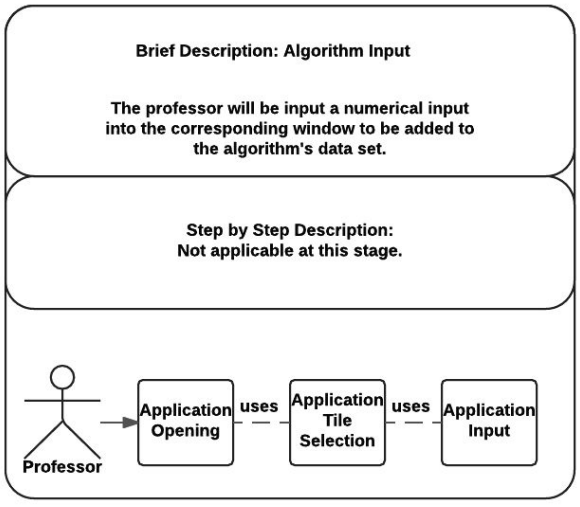


Figure 3

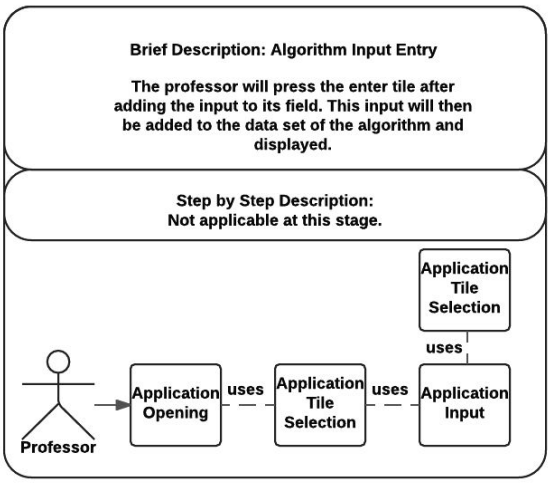


Figure 4

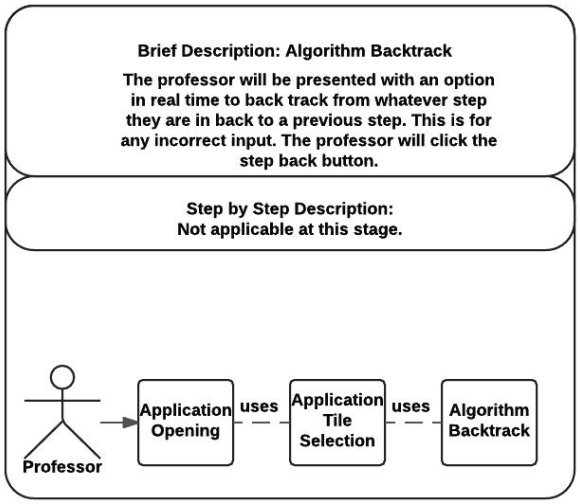


Figure 5

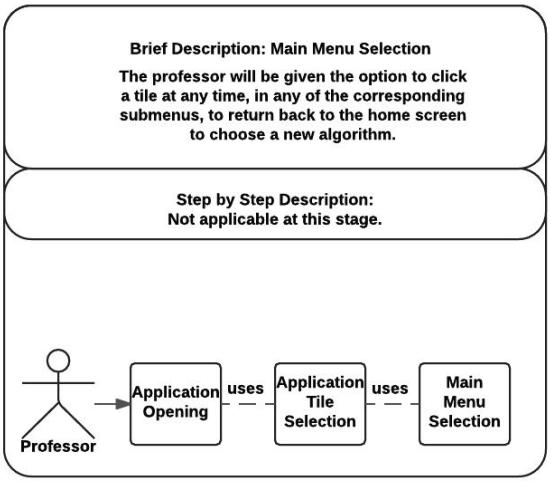


Figure 6

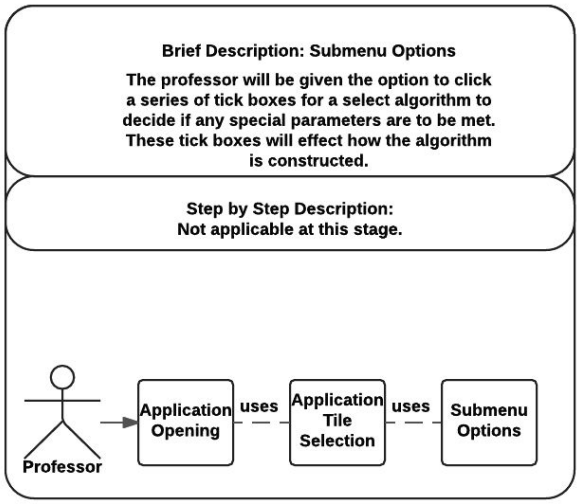
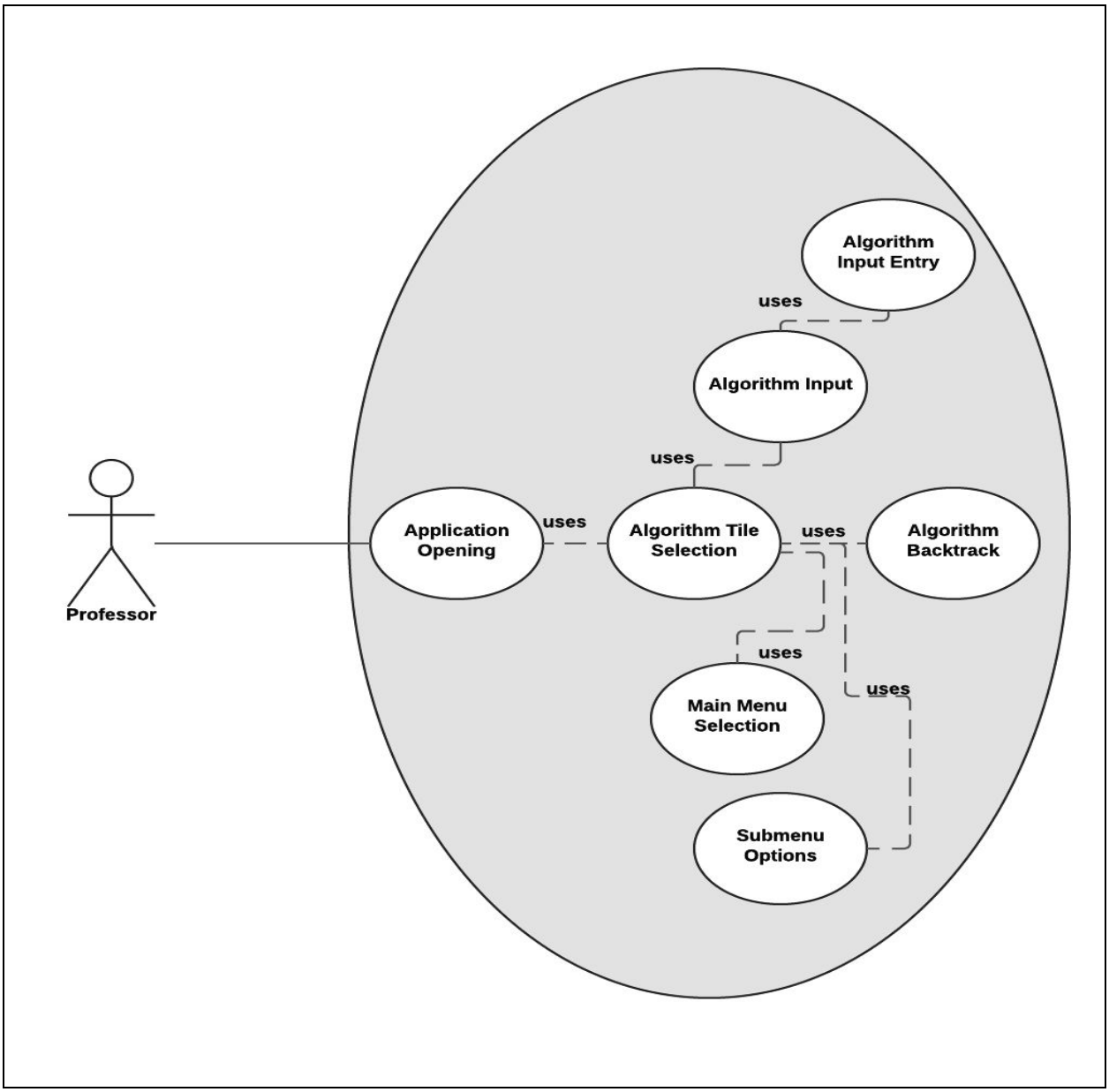


Figure 7

Final Requirements

➢ **Case:**



➢ **Brief Summary:**

After the iterations we end up with a much more accurate set of requirements (Final Requirements) divided into two phases: the *Menu Selection Phase* and the *Iteration Phase.*

The *Menu Selection* phase encompasses the choosing of what algorithm is going to be ran according to the professor's choice. This phase consists of a different menu than the sub menus, and will function as the "home" of the application. All return to main menu options will return the professor to this phase.

The *Iteration* phase encompasses the actions that the professor will be taking to implement the chosen algorithms from the application. This includes data input, submenu options, and returning to the main menu.

➢ **Metrics**

- During the development of this software certain metrics are going to be required to ensure that development runs smoothly and produces a quality product. In order to establish the metrics we need to look at some key factors about the development of this project.

  Volatility: In order to see the evolution of the application and being able to fully refine its design, we will be keeping track of how frequently the requirements change in order to determine the rate on which this team converges on requirements.

Efficiency: Efficiency is key in the development of any software; it ensure that our efforts are being spent in a productive manner through the developmental process. In order to achieve the expected efficiency, we will be keeping track of which part of the effort is being spent on reworking the requirements and the analysis workflows. By analyzing the efficiency of the team using Intermediate Cocomo, we are able to realize the effort we are spending on the project as a whole.

Cohesion: Cohesion is imperative as with most software products. In order to determine this quality we will be responsible for measuring the responsibilities each of our classes possess, thus realizing which classes will need to be split or refactored to be able to achieve a greater performance.

Size: In order for us to develop the software product in a manner conducive with being as lightweight as possible we need to have a keen understanding on just how big the software is. During the development we are going to be keeping track of the LOC. Which we will then utilize to calculate the errors per KLOC, and defect per KLOC. Evaluating these will lead us towards a better quality process.

Appended Material

Non-execution based testing:

Our team will utilize the inspection process for our non-execution based testing. As opposed to walkthroughs, this will allow us to track our metric of size, as well as implement a metric to determine the fault density. Skylar will act as moderator, Jorge as the recorder, and Mitchell as the reader. The moderator is the leader of the inspection, and is in charge of planning the inspection and coordinating it. The recorder is in charge of documenting all defects that are found during the inspection. The reader reads through the documents, one item at a time, cross examining all the defects the other inspectors have pointed out.

Using the inspection process, we discovered several defects that were causing some errors and buttons to not work properly. Upon discovery, these defects were taken care of and fixed to increase efficiency.

Metrics:

In terms of volatility for this document, we kept a record for how frequently a change was implemented by one of the team members following completion of the document. Averaging at roughly 23.5 changes a week, with 141 changes total.

CODENAME: JIGGLYPUFF

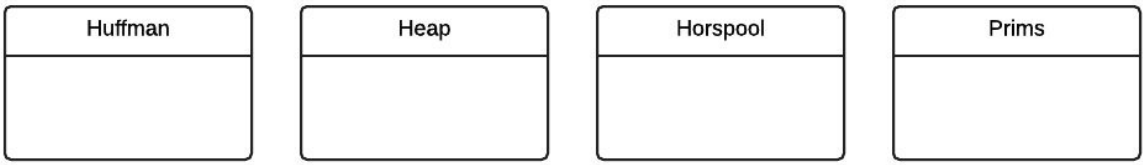
Analysis Document

Skylar Fritzky
Mitchell Shaw
Jorge Koifman

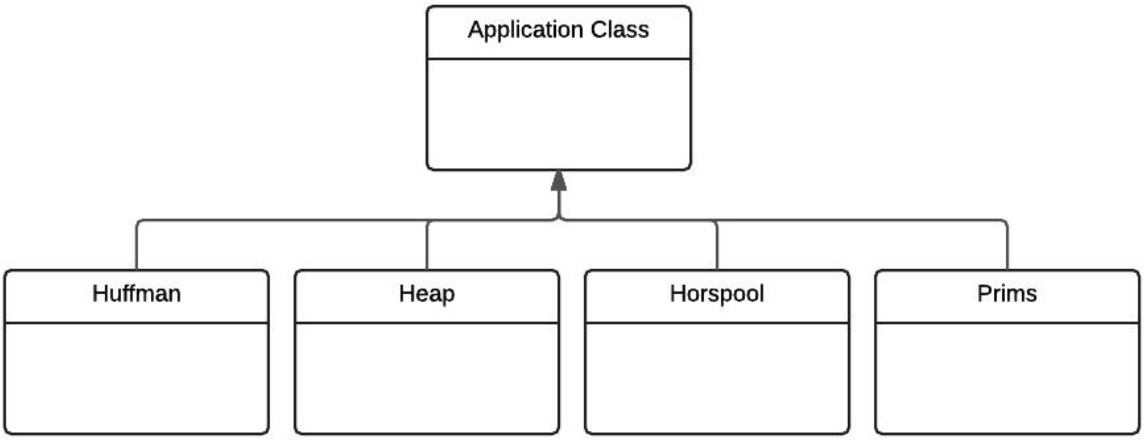
Appendix

Class Model

Phase 1



Phase 2



Phase 3

This is the class diagram for our product, we have chosen **Provider**, **Services**, and **Member** to all be entity classes. The information of these classes needs to be saved long term in our system so that we are able to maintain all of the users of our services. These classes are all a subclass of our superclass **Choco Application**, that maintains all of the interactions between the *providers*, *members*, and the *Data Center*.

Class Extraction

Scenario

A. Application Opening
    a. Professor A locates the .exe file
    b. Professor A clicks the file and opens the application.
B. Algorithm Tile Selection
    a. Professor A decides which algorithm that he would like to demonstrate.
    b. Professor A clicks the corresponding tile to what he would like to open.
    c. The application redirects Professor A to the correct submenu screen.
C. Algorithm Input
    a. Professor A is in the corresponding subscreen for the algorithm selected.
    b. Professor A inputs his data into the data field for algorithm implementation.
D. Algorithm Input Entry
    a. Professor A has entered an entry into the data field.
    b. Professor A clicks the “Enter” button and the application saves the entry to a field.
E. Algorithm Backtrack
    a. Professor A would like to step back in the submenu due to an entry error.
    b. Professor A clicks the “Back” button.
    c. The algorithm steps back one full step and is back at the previous state of the algorithm.
F. Main Menu Selection
    a. Professor A has completed demonstrating his algorithm and would like to backtrack to the previous screen.
    b. Professor A clicks the “Main Menu” button and returns back to the “Algorithm Tile Selection” screen.

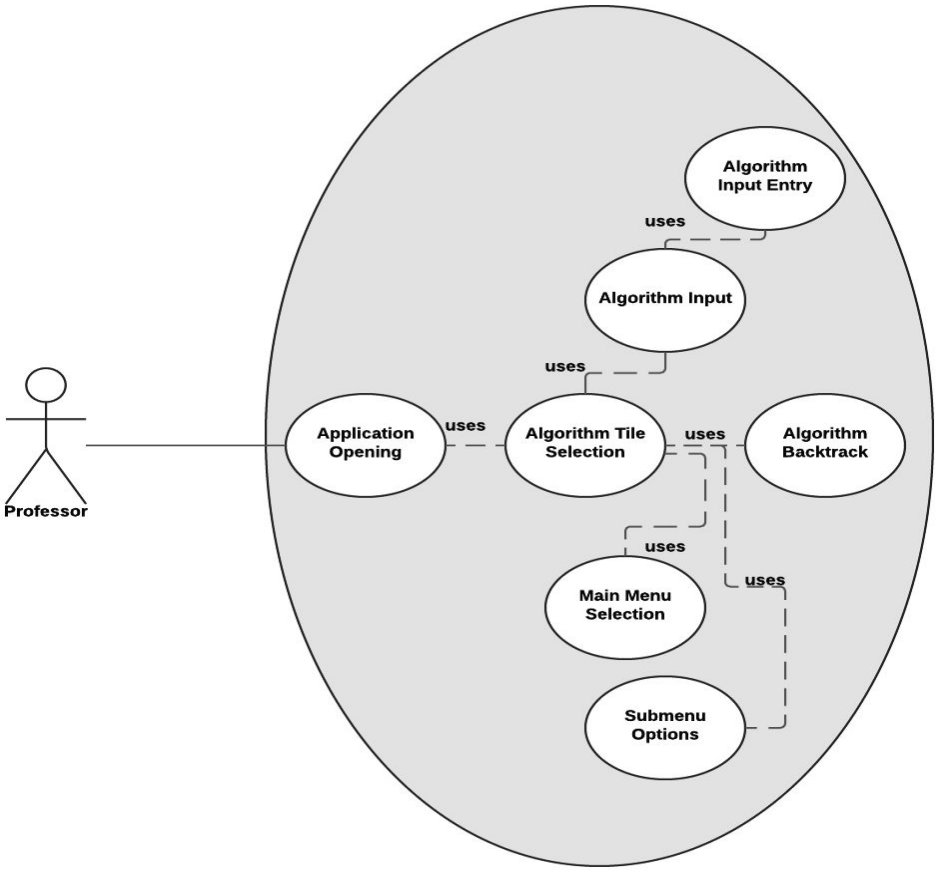
Exception Scenario

A. Application Opening
    a. Professor A cannot locate the .exe file
    b. The program cannot be open.
B. Algorithm Input

a. Professor A enters the wrong input or an illegal entry into the field.
b. An exception is thrown to catch this and Professor A must re-enter the entry.

C. Algorithm Input Entry
a. Professor A has entered an entry into the data field.
b. Professor A clicks the “Enter” button and the application saves the entry to a field.

Scenario Diagram



Noun Extraction

We have created a simple paragraph that describes what the application does, here we will extract our nouns that will become entity classes:

- The software consists of a series of buttons that we will call Tiles. These Tiles consist of Heap Construction, Horspool's Algorithm, Huffman's Algorithm, and Prim's Algorithm. Upon Tile selection, the main menu will navigate to the corresponding tile. If Heap Construction is chosen, a submenu immediately will be displayed, asking for parameters of how the heap will be constructed (top-down or bottom-up). The heap will then be constructed logically using this parameter, step by step as data is entered. Horspool's algorithm will display a submenu as well immediately asking for two input strings, the string we are searching and the keyword we are searching for. The algorithm will then display step by step using indentation for each logical step and a shift table will be displayed on the screen for each given character. Huffman's Algorithm will present another submenu, this time asking for a set of values and a frequency that corresponds to each value. The algorithm will then step through the tree and output the encoding for the final tree's values in the form of a table. Prim's is the only exception to our "solver", Prim's algorithm will prompt the user to show one of three solutions to a predetermined graph and show how to logically step through each one. This is simply for execution purposes, as creating this graph to a user's specifications would present a larger problem than this application seeks to solve. On all of the submenu screens during visualization, the Professor will have the ability to step back one step of the visualization in case a student does not understand one step. At any time, the Professor may cancel out of visualization to return to the main menu. At the main menu screen the Professor will have the option to select a new algorithm or exit the application.

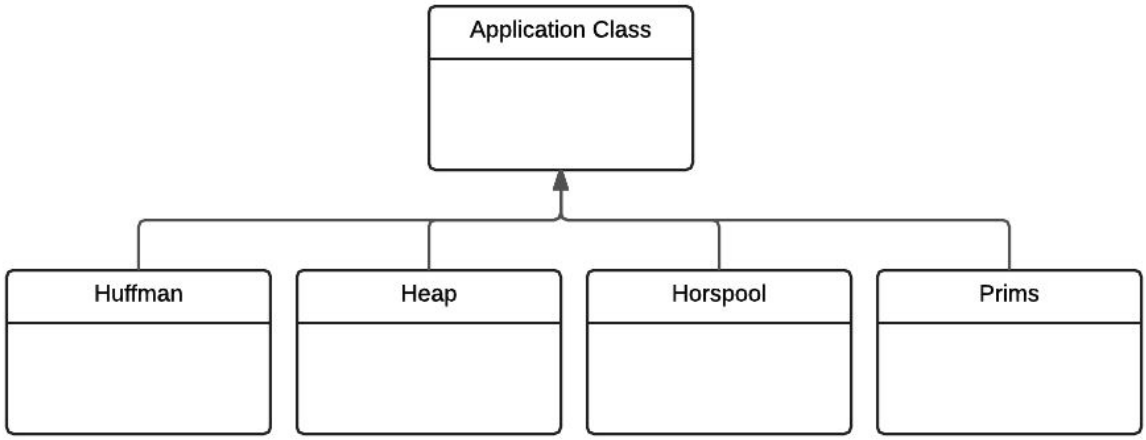
Noun extraction is occurring in the step. We will bold all important nouns regarding class extraction as follows:

- The software consists of a series of buttons that we will call **Tiles**. These Tiles consist of **Heap Construction**, **Horspool's Algorithm**, **Huffman's Algorithm**, and **Prim's Algorithm**. Upon **Tile** selection, the **main menu** will navigate to the corresponding **tile**. If **Heap Construction** is chosen, a **submenu** immediately will be displayed, asking for

parameters of how the **heap** will be constructed (top-down or bottom-up). The **heap** will then be constructed logically using this parameter, step by step as data is entered. **Horspool's algorithm** will display a **submenu** as well immediately asking for two input strings, the **string** we are searching and the **keyword** we are searching for. The algorithm will then display step by step using indentation for each logical step and a **shift table** will be displayed on the screen for each given character. **Huffman's Algorithm** will present another **submenu**, this time asking for a set of **values** and a **frequency** that corresponds to each **value**. The algorithm will then step through the **tree** and output the **encoding** for the final tree's values in the form of a **table**. **Prim's** is the only exception to our "solver", **Prim's Algorithm** will prompt the user to show one of three solutions to a predetermined **graph** and show how to logically step through each one. This is simply for execution purposes, as creating this graph to a user's specifications would present a larger problem than this application seeks to solve. On all of the **submenu** screens during visualization, the Professor will have the ability to step back one **step** of the visualization in case a student does not understand one **step**. At any time, the Professor may cancel out of visualization to return to the **main menu**. At the **main menu screen** the Professor will have the option to select a **new algorithm** or exit the **application**.

Entity

- Entity Diagram



- Entity Description
    - We notice in the Scenario that the Professor is interacting with the application by opening the .exe file. These are all the nouns of the algorithms associated with the application software that should be individual entity classes.

- Entity CRC Cards

Huffman	Huffman Submenu, Huffman Solver
• Huffman will act as the overall container for our Huffman algorithm, it will be responsible for showing the visualization of Huffman's algorithm, This class will represent a number based in a tree in it's binary representation based on the Huffman's algorithm.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

Heap	Heap Submenu, Heap Solver
• Heap will act as the overall container for our Heap Construction algorithm, it will be responsible for showing the visualization of the heap we construct. This class will represent a heap based on constraints created in the subclasses, Heap Submenu and Heap Solver.	• Main Menu Button • Exit Button • Step Back Button • UI Application

Horspool	Horspool Submenu, Horspool Solver
• Horspool will act as the overall container for the Horspool algorithm, it will be responsible for showing the visualization of Horspool's algorithm, This class will represent a string and a substring that will be searched within the main string. It will show the visualization of the shift's of the substring and a shift table for each letter corresponding to the substring. It will use constraints passed in the subclasses, Horspool Submenu and Horspool Solver.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

Prims	Prims Submenu, Prims Solver
• Prims will act as the overall container for our Prim's algorithm, it will be responsible for showing the pre-created examples based on constraints passed in the subclasses, Prim's Submenu and Prim's Solver, These examples are pre-created to only demonstrate a basic foundation of movement in the algorithm.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

Boundary

- Boundary Description
    - These are the initial boundary classes, a boundary class is based on the interactions between the software and the user. The following classes are considered boundary classes in our application: **UI, Main Menu Button, Exit Button, Heap Submenu, Horspool Submenu, Huffman Submenu, Prims Submenu, Step Back.** All of the events from the event loop can be found here, as the event loop is the visual representation of all interactions from the user to the software. Our boundary classes encompass the classes that we believe the user will be using regularly while interacting with the application. **UI** is the basic GUI that will access as the portal to all other menus. **Main Menu Button** is the return home button that will allow a user to click a button and return home. **Exit Button** is the close button that acts as a secondary resource for the user to close the application. **Step Back Button** is the undo button for the application, the user will click this button if they would like to backtrack during algorithm visualization. **Heap Submenu, Horspool Submenu, Huffman Submenu, and Prims Submenu** are the acting submenu options for the application. These sub menus will act as parameter in take where the user will input the data that they want to see visualized. Each corresponding sub menu will have parameters to be set in relation to how the algorithm is constructed.

- Boundary CRC Cards

UI	Application Main Menu Button, Exit Button, Step Back Button
• UI will act as the graphical user interface for the user, it will also act as the container for the buttons to navigate to appropriate screens. This class is the direct line of communication from user to system.	• Exit Button • Application Class • Huffman • Heap • Horspool • Prims

Main Menu Button	UI, Application
• Main Menu Button will serve as the return to home button for the application, at any time the user may press this button to exit what they are doing and return back to the initial start screen.	• Application Class • Huffman • Heap • Horspool • Prims • Huffman Submenu • Heap Submenu • UI • Horspool Submenu • Prims Submenu • Huffman Solver • Heap Solver • Horspool Solver • Prims Solver

Exit Button	Application
• Exit Button will serve as the quit application button, this button is available only on the main screen and will terminate the application,	• Application Class • Huffman • Heap • Horspool • Prims • Huffman Submenu • Heap Submenu • UI • Horspool Submenu • Prims Submenu • Huffman Solver • Heap Solver • Horspool Solver • Prims Solver

Huffman Submenu	Huffman Huffman Solver
• Huffman Submenu will act as the class to determine constraints for the Huffman class. It will be used to determine the frequency of a character and a set of characters.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

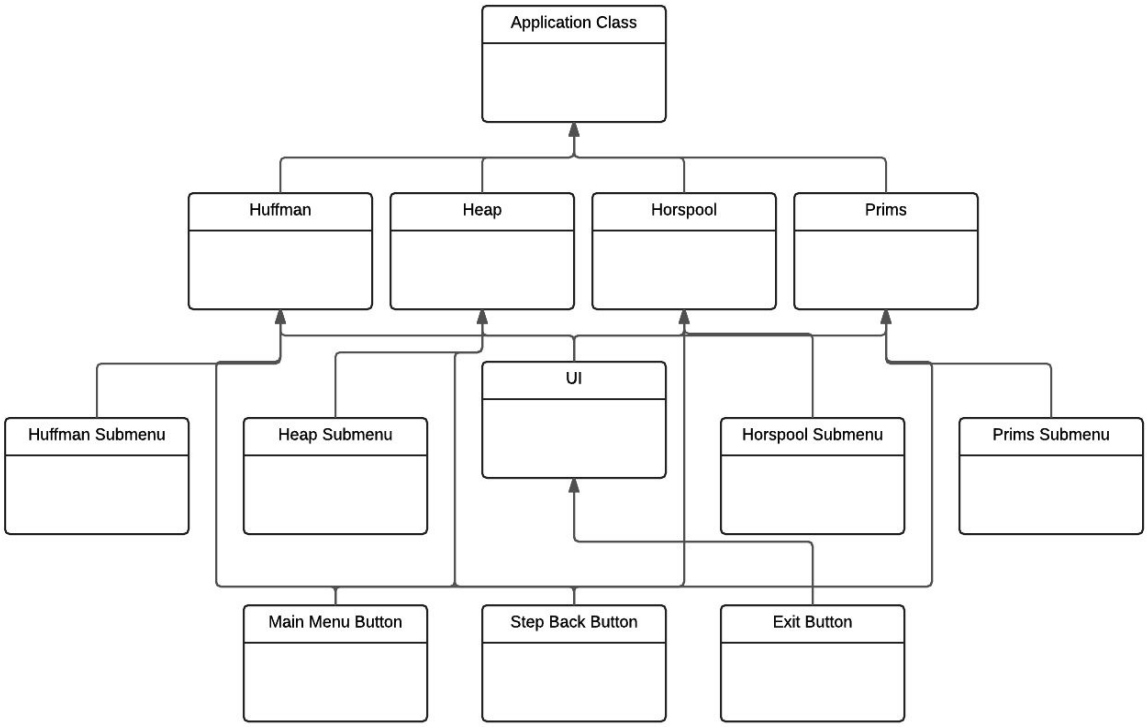
Heap Submenu	Heap Heap Solver
• Heap Submenu will act as the constraint for the superclass Heap, this class will set the constraints for Heap to determine if the heap will be created using a top-down or bottom up approach.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

Horspool Submenu	Horspool Horspool Solver
• Horspool submenu will act as the constraint for Horspool, it will take in the search string and a substring to search for within that string.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

Prims Submenu	Prims Prims Solver
• Prims submenu is the container for the constraints for Prims, it will present simple menu of choices to show an example based on the selection chosen.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

Step Back Button	Application
• Step Back Button will act as a undo button for the application and will step back one full step in the regarded algorithm.	• Application Class • Huffman • Heap • Horspool • Prims • Huffman Submenu • Heap Submenu • UI • Horspool Submenu • Prims Submenu • Huffman Solver • Heap Solver • Horspool Solver • Prims Solver

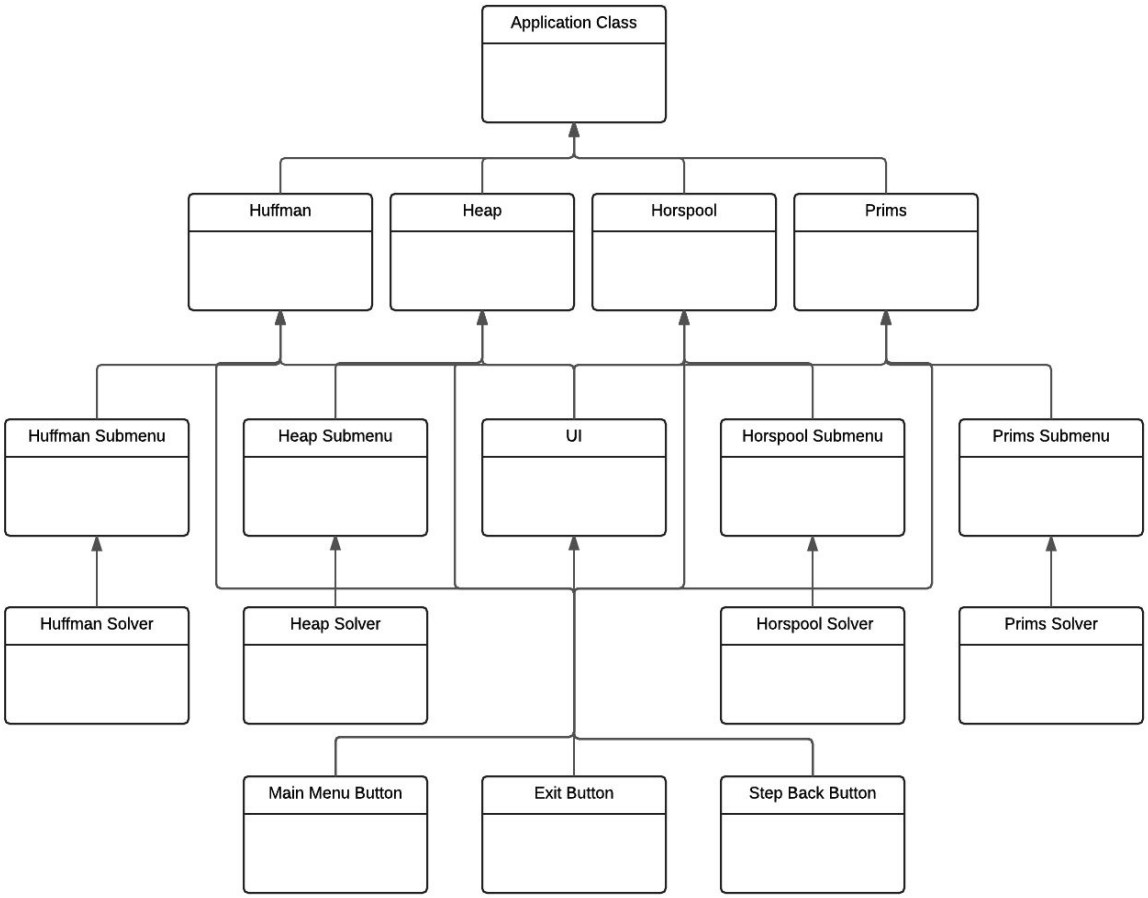
- Boundary Diagram



Control

- Control Description
    - The Control classes focus on computational power in our application. These classes are our algorithm solver's. **Huffman Solver, Heap Solver, Horspool Solver, and Prims Solver**. The logic behind these algorithms is presented and brought into the superclass for visualization to the user. Each submenu will follow independent logic of one another.

- Control Diagram



- CRC Cards

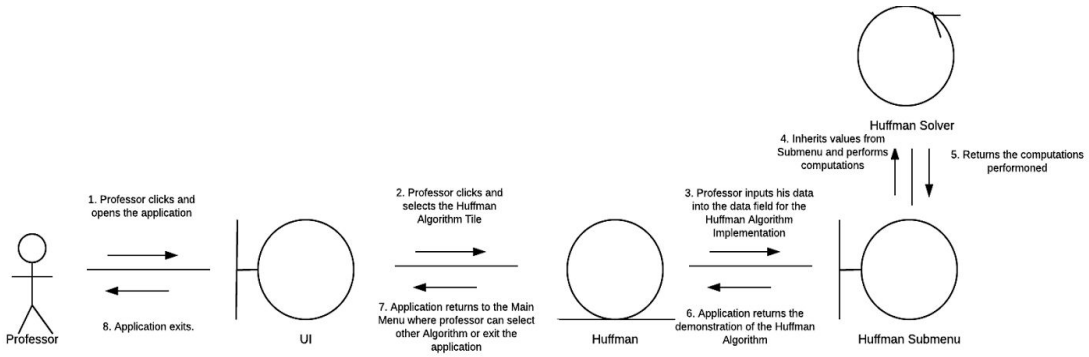
Heap Solver	Heap Submenu, Heap
• Heap Solver will act as the logic for the Heap class, and will take constraints from the Heap Submenu to accurately provide the logic for heap construction.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

Horspool Solver	Horspool Submenu, Horspool
• Horspool Solver will act as the logic for the Horspool class, it will take constraints and values from the Horspool Submenu class and perform substring matching based on the Horspool algorithm.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

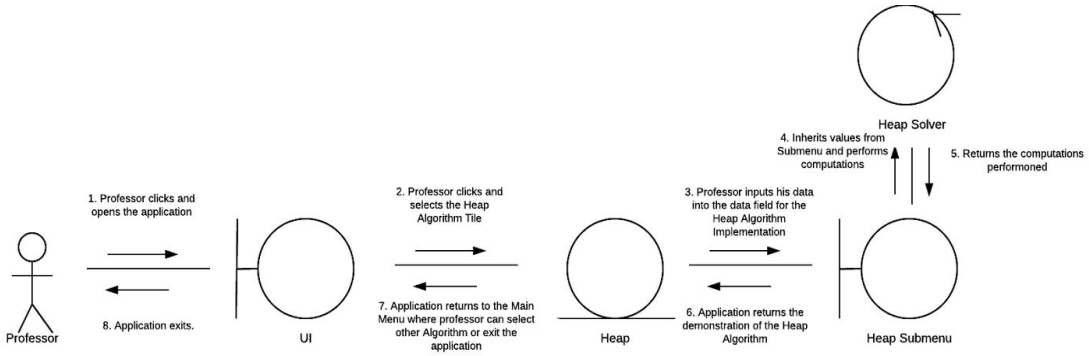
Huffman Solver	Huffman Submenu, Huffman
• Huffman Solver is the logic behind the Huffman class, it will inherit the values passed in the Huffman submenu and perform the general computation based on the Huffman algorithm.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

Prims Solver	Prims Submenu, Prims
• Prims Solver will serve as the logic behind Prims, using the selected example from Prims Submenu, it will show the correct path using Prim's algorithm.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

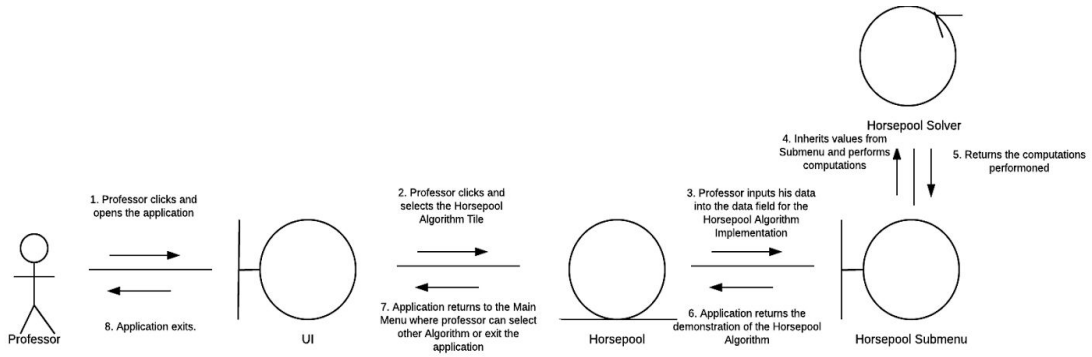
Use-Case Realization



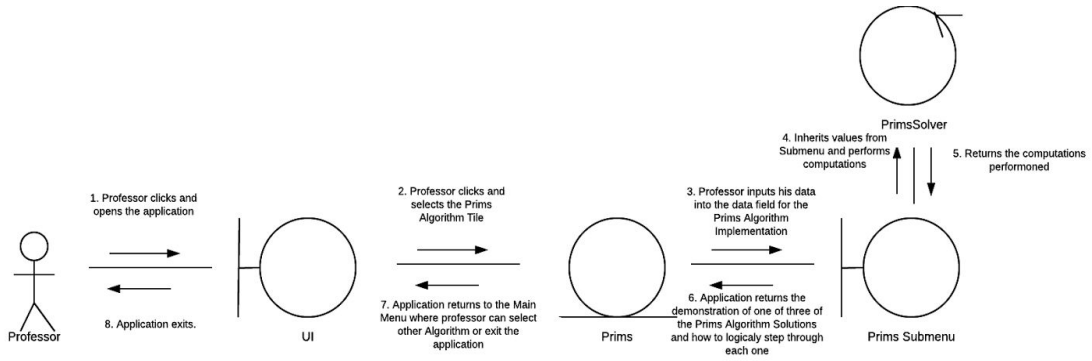
The use case realization of the *Huffman Algorithm,* this communication diagram shows the flow of communication among two different boundary classes, **UI class** and **Huffman Submenu class,** one entity class, **Huffman,** and one control class, **Huffman Solver.**



The use case realization of the *Heap Algorithm*, this communication diagram shows the flow of communication among two different boundary classes, **UI class** and **Heap Submenu class,** one entity class, **Heap,** and one control class, **Heap Solver.**



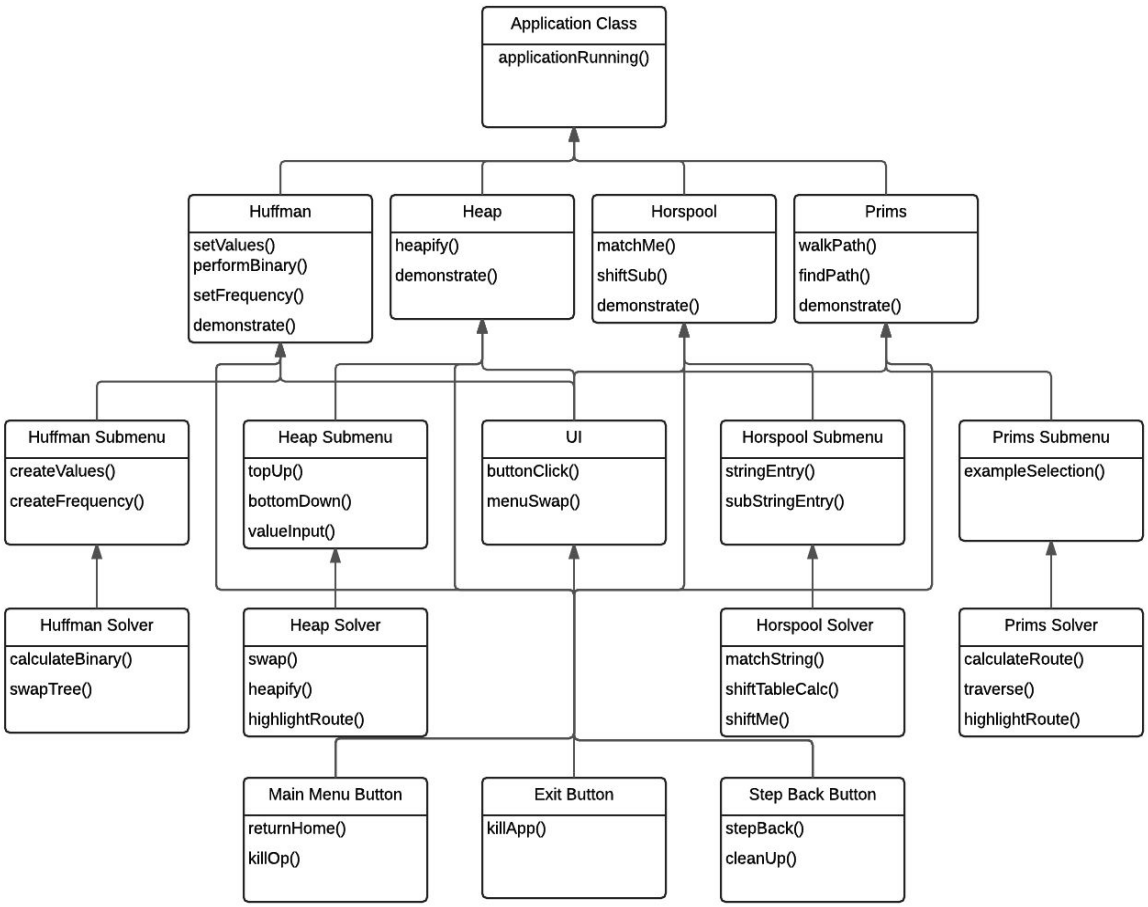
The use case realization of the *Horsepool Algorithm*, this communication diagram shows the flow of communication among two different boundary classes, **UI class** and **Horsepool Submenu class,** one entity class, **Horsepool,** and one control class, **Horsepool Solver**



The use case realization of the *Prims Algorithm*, this communication diagram shows the flow of communication among two different boundary classes, **UI class** and **Prims Submenu class,** one entity class, **Prims,** and one control class, **Prims Solver**.

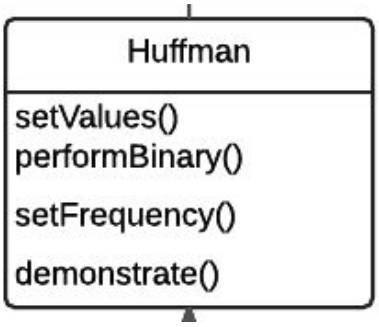
Final Classes

Final Class Diagram



Final Entity Classes

- Huffman Class
    - Class Diagram



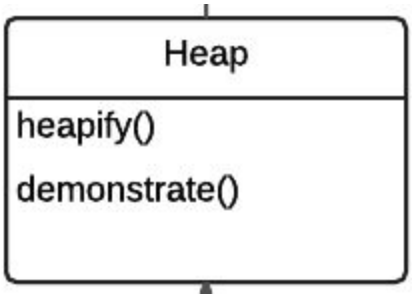
- Interaction Diagrams

  See use case communication diagrams

- CRC Card

Huffman	Huffman Submenu, Huffman Solver
• Huffman will act as the overall container for our Huffman algorithm, it will be responsible for showing the visualization of Huffman's algorithm, This class will represent a number based in a tree in it's binary representation based on the Huffman's algorithm.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

- Heap Class
  - Class Diagram



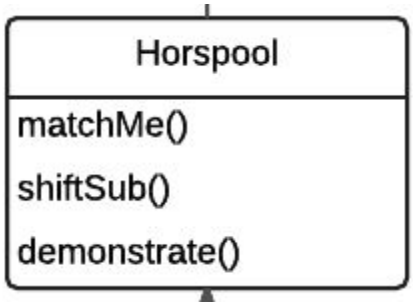
  - Interaction Diagrams

    See use case communication diagrams

  - CRC Card

Heap	Heap Submenu, Heap Solver
• Heap will act as the overall container for our Heap Construction algorithm, it will be responsible for showing the visualization of the heap we construct. This class will represent a heap based on constraints created in the subclasses, Heap Submenu and Heap Solver.	• Main Menu Button • Exit Button • Step Back Button • UI Application

- Horspool Class
    - Class Diagram



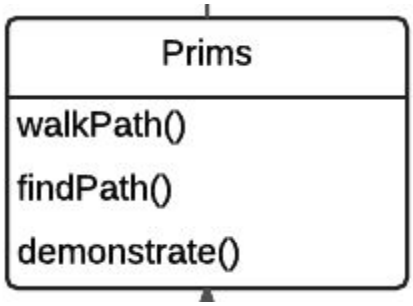
    - Interaction Diagram

See use case communication diagrams

    - CRC Card

Horspool Horspool Submenu, Horspool Solver	
• Horspool will act as the overall container for the Horspool algorithm, it will be responsible for showing the visualization of Horspool's algorithm, This class will represent a string and a substring that will be searched within the main string. It will show the visualization of the shift's of the substring and a shift table for each letter corresponding to the substring. It will use constraints passed in the subclasses, Horspool Submenu and Horspool Solver.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

- Prims Class
    - Class Diagram



    - Interaction Diagram

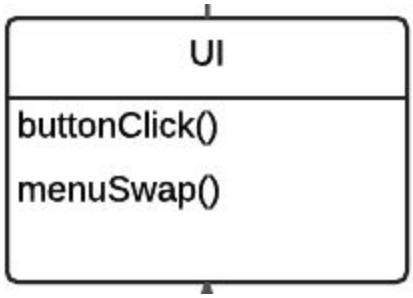
See use case communication diagrams

    - CRC Card

Prims	Prims Submenu, Prims Solver
• Prims will act as the overall container for our Prim's algorithm, it will be responsible for showing the pre-created examples based on constraints passed in the subclasses, Prim's Submenu and Prim's Solver, These examples are pre-created to only demonstrate a basic foundation of movement in the algorithm.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

Final Boundary Classes

- UI Class
    - Class Diagram



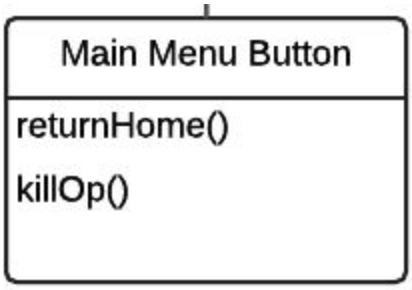
- Interaction Diagram

    See use case communication diagrams

- CRC Cards

UI	Application Main Menu Button, Exit Button, Step Back Button
• UI will act as the graphical user interface for the user, it will also act as the container for the buttons to navigate to appropriate screens. This class is the direct line of communication from user to system.	• Exit Button • Application Class • Huffman • Heap • Horspool • Prims

- Main Menu Button Class
    - Class Diagram



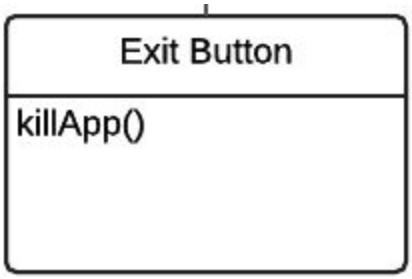
    - Interaction Diagram

      See use case communication diagrams

    - CRC Cards

Main Menu Button	UI, Application
• Main Menu Button will serve as the return to home button for the application, at any time the user may press this button to exit what they are doing and return back to the initial start screen.	• Application Class • Huffman • Heap • Horspool • Prims • Huffman Submenu • Heap Submenu • UI • Horspool Submenu • Prims Submenu • Huffman Solver • Heap Solver • Horspool Solver • Prims Solver

- Exit Button Class
    - Class Diagram



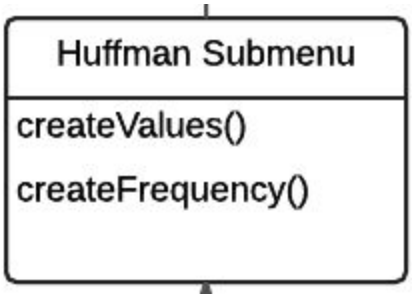
- Interaction Diagram

  See use case communication diagrams

- CRC Card

Exit Button	Application
• Exit Button will serve as the quit application button, this button is available only on the main screen and will terminate the application,	• Application Class • Huffman • Heap • Horspool • Prims • Huffman Submenu • Heap Submenu • UI • Horspool Submenu • Prims Submenu • Huffman Solver • Heap Solver • Horspool Solver • Prims Solver

- Huffman Submenu Class
  - Class Diagram



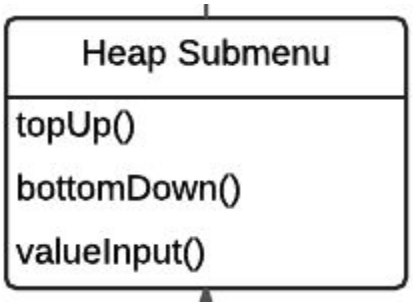
  - Interaction Diagrams

    See use case communication diagrams

  - CRC Card

Huffman Submenu	Huffman Huffman Solver
• Huffman Submenu will act as the class to determine constraints for the Huffman class. It will be used to determine the frequency of a character and a set of characters.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

- Heap Submenu Class
    - Class Diagram



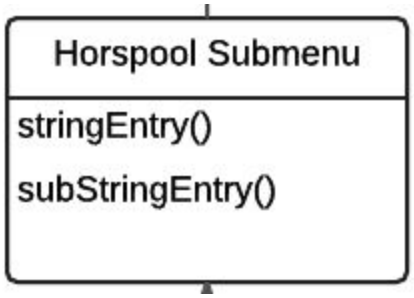
    - Interaction Diagram

See use case communication diagrams

    - CRC Card

Heap Submenu	Heap Heap Solver
• Heap Submenu will act as the constraint for the superclass Heap, this class will set the constraints for Heap to determine if the heap will be created using a top-down or bottom up approach.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

- Horspool Submenu Class
    - Class Diagram



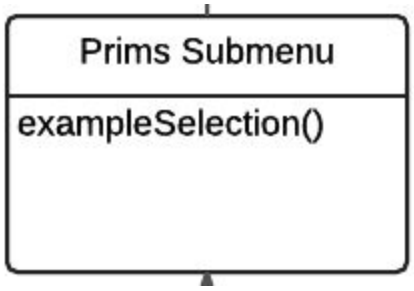
- Interaction Diagram

See use case communication diagrams

- CRC Card

Horspool Submenu	Horspool Horspool Solver
• Horspool submenu will act as the constraint for Horspool, it will take in the search string and a substring to search for within that string.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

- Prims Submenu Class
  - Class Diagram



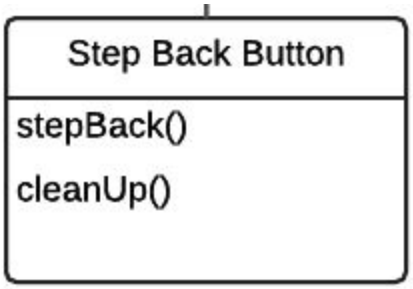
  - Interaction Diagram

See use case communication diagrams

  - CRC Card

Prims Submenu	Prims Prims Solver
• Prims submenu is the container for the constraints for Prims, it will present simple menu of choices to show an example based on the selection chosen.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

- Step Back Button Class
    - Class Diagram



    - Interaction Diagram

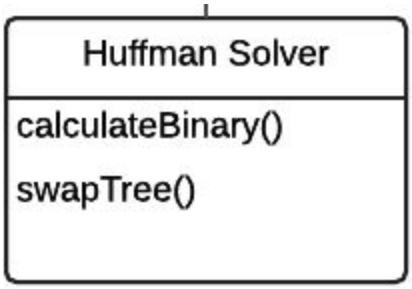
See use case communication diagrams

    - CRC Card

Step Back Button	Application
• Step Back Button will act as a undo button for the application and will step back one full step in the regarded algorithm.	• Application Class • Huffman • Heap • Horspool • Prims • Huffman Submenu • Heap Submenu • UI • Horspool Submenu • Prims Submenu • Huffman Solver • Heap Solver • Horspool Solver • Prims Solver

Final Control Classes

- Huffman Solver Class
    - Class Diagram



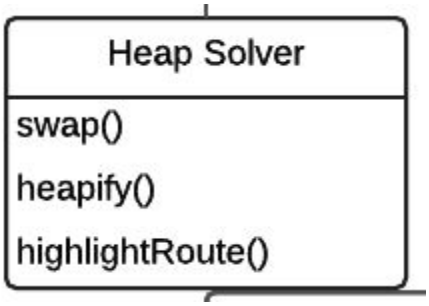
    - Interaction Diagram

See use case communication diagrams

    - CRC Card

Huffman Solver	Huffman Submenu, Huffman
• Huffman Solver is the logic behind the Huffman class, it will inherit the values passed in the Huffman submenu and perform the general computation based on the Huffman algorithm.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

- Heap Solver Class
    - Class Diagram



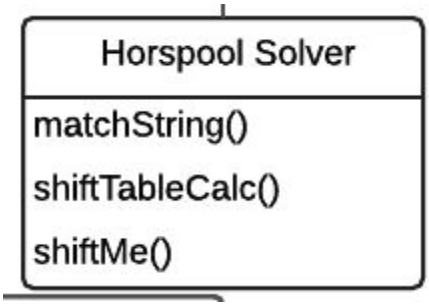
    - Interaction Diagram

See use case communication diagrams

    - CRC Card

Heap Solver	Heap Submenu, Heap
• Heap Solver will act as the logic for the Heap class, and will take constraints from the Heap Submenu to accurately provide the logic for heap construction.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

- Horspool Solver Class
    - Class Diagram



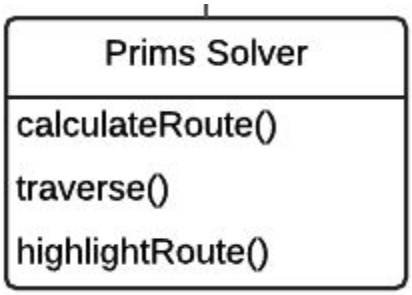
    - Interaction Diagram

See use case communication diagrams

    - CRC Card

Horspool Solver	Horspool Submenu, Horspool
• Horspool Solver will act as the logic for the Horspool class, it will take constraints and values from the Horspool Submenu class and perform substring matching based on the Horspool algorithm.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

- Prims Solver Class
    - Class Diagram



    - Interaction Diagram

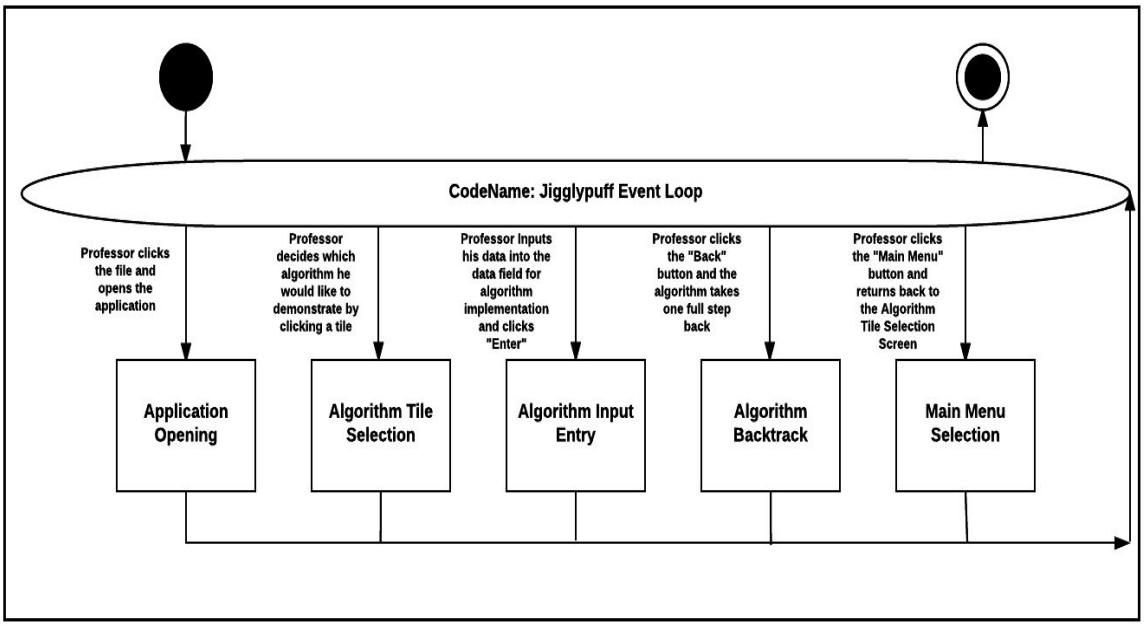
See use case communication diagrams

    - CRC Card

Prims Solver	Prims Submenu, Prims
• Prims Solver will serve as the logic behind Prims, using the selected example from Prims Submenu, it will show the correct path using Prim's algorithm.	• Main Menu Button • Exit Button • Step Back Button • UI • Application

Dynamic Model

Diagram:



Description:

This is the **Codename: Jigglypuff Event Loop,** here all actions performed on the software by the user are identified. The professor interacts with the application in the following ways. The professor initiates the event loop by opening the application with the .exe file. **Algorithm Tile Selection:** The professor then proceeds to decide and select which algorithm to demonstrate by clicking the corresponding tile. **Algorithm Input Entry:** The professor inputs the data into the data field for the algorithm implementation and clicks “Enter” which makes the algorithm run. **Algorithm Backtrack:** The professor clicks the “Back” button and the algorithm takes a full step back and is back at the previous state of the algorithm. **Main Menu Selection:** The professor clicks the “Main Menu” button and returns to the Algorithm selection Screen where he can choose another algorithm to demonstrate or exit.

Metrics

To further examine and calculate the significance of the precision of our analysis we used methods and time logs to observe how much work, time and effort were put into each section of the document. Additionally, we cross referenced all of our old documents with the new ones to further evaluate the evolution of the understanding of the application from initial drafts to even the newest aspects of the documents. Timelogs date as far back as the 31st of January of this year(2017), and the newest ones dates to the most recent present date, February 21st, 2017. Requirements document ideas we had have further been explained more meticulously and observed to provide a better understanding further down the line to give a more precise plan of action for the application.

Appended Material

Execution Based Testing:

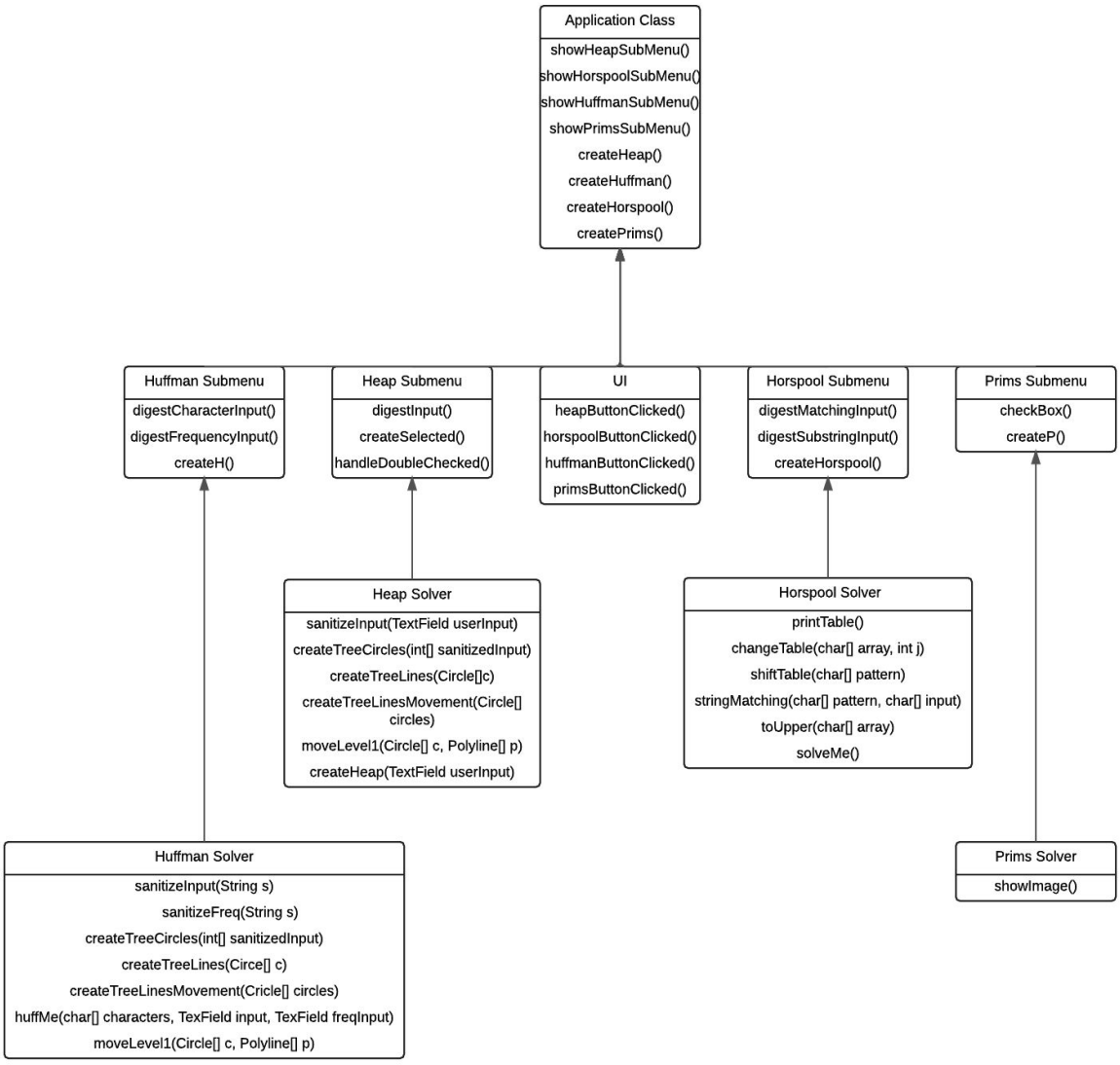
For our testing we will be using two types White-box testing methods. Statement coverage will be the first, which is basically a means to include extreme outliers and sanitize them as they are included from user input because it involves the execution of all the statements at least once in the source code.. Branch coverage will be the latter, which is simply testing statements to determine if all lines are executed upon given input, due to the fact that branch coverage aims to ensure that each one of the possible branches from each decision point is executed at least once and thereby ensuring that all reachable code is executed including every true and false decisions. The testing pattern includes Jorge testing Skylar's code, Skylar testing Mitchell's code, and Mitchell will test both of his members' code.

CODENAME: JIGGLYPUFF

Design Document

Skylar Fritzky
Mitchell Shaw
Jorge Koifman

Class diagram



Huffman Submenu: This subclass of Huffman acts to determine constraints for the Huffman class. I will be used determine the frequency of a character and a set of characters.

```
package sample.huffmanSubmenuController;

import sample.Main;
import javafx.fxml.*;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.TextField;
import javafx.scene.control.Label;
import jdk.internal.util.xml.impl.Input;
import sample.Main;
import javafx.application.*;
import javafx.fxml.*;
import javafx.scene.*;
import javafx.scene.layout.BorderPane;
import javafx.stage.*;

import javafx.fxml.*;

public class huffmanSubmenuController {
    private Main main;
    private TextField cInput;
    private TextField fInput;
    private Button createHuffman;

    @FXML
    private void digestCharacterInput(){
        String characters = cInput.getText();
    }
    @FXML
    private void digestFrequencyInput(){
        String frequency = fInput.getText();
    }
    @FXML
    private void createH(){
        main.createHuffman();
    }
}
```

Huffman Solver: This subclass of Huffman will act as the logic for the Huffman class, and it will inherit values passed in the submenu class and perform the general calculations based on the Huffman algorithm.

```
package sample.huffmanSubmenuController;

import javafx.animation.PathTransition;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.scene.shape.Polyline;

import java.awt.*;
import java.util.Arrays;

/**
 * Created by Mitchell on 3/27/2017.
 */
public class huffmanSolver{

   private char[] sanitizeInput(String s){
      char[] charArray = s.toCharArray();
      return charArray;
   }
   private int[] sanitizeFreq(String s){
      char[] charArray = s.toCharArray();
      for(int x=0; x<sanitize;x++)
      {
         char c= charArray[i];
         array[i]= Character.getNumericValue(c);
      }
      return array;
   }
   private void createTreeCircles(int[] sanitizedInput) {
      Circle[] circles = new Circle[sanitizedInput.length];
      double shiftX= 100;
      double shiftY= 100;

      for (int x = 0; x < circles.length; i++) {
         circles[x] = new Circle(45, shiftX, shiftY); //root node is at top
         if (x % 2 == 0) {
```

```
            shiftX some amount
        } else {
            shiftX some other amount for the right child
        }

    }}
private void createTreeLines(Circle[] c){
    Line[] lines = new Line[c.length*2];
    double startX= 100;
    double startY= 100;
    double stopX= 200;
    double stopY= 200;
    for (int x = 0; x < lines.length; i++) {
        lines[x] = new lines(startX,startY,stopX,stopY); //root node is at top
        if(x%2 == 0)
        {
            shiftX some amount
        }
        else
        {
            shiftX some other amount for the right child
        }




    }
}
private void createTreeLinesMovement(Circle[] circles){
    if(circles.length%2 ==0)
    {
        Polyline[] lines= new Polyline[circles.length*2]; //twice as many lines as circles if even
    }
    else
    {
        int linesTotal= circles.length*2;
        linesTotal=linesTotal-1;
        Polyline[] lines= new Polyline[linesTotal];
    }
}

private void moveLevel1(Circle[] c,Polyline[] p) {
    PathTransition transition1 = new PathTransition();
    transition1.setNode(circle[i]);
```

```
        transition1.setDuration(2);
        transition1.setPath(p[0]);
    }
    //repeat above method for the other shapes
    private void huffMe(char[] characters, TextField input, TextField freqInput)
    {
        String userInput = input.getText();
        String frequency = freqInput.getText();
        int[] myFreq = sanitizeFreq(frequency);
        char[] myChars= sanitizeInput(userInput);
        int[] totalBits=new int[myFreq.length];
        String[] code = new String["000","001","010","011","100","101","110","111"];
        int k;

        for(int i=0; x<myChars.length;i++)
        {
            k=code[i].length;
            totalBits[i]=myFreq[i]*k;
        }
        //sort and find lowest frequency
        Arrays.sort(myFreq);
        //match them with their respective string
        int parentValue;
        for(int i=0; i<myFreq.length;i+2)
        {
            parentValue=myFreq[i]+myFreq[i+1];
        }
    }
}
```

Heap Submenu: This subclass of Heap acts to create the constraints of the Heap class as well as insert the inputs. The constraints include top up and bottom down approaches to solving the Heap Construction algorithm.

```
package sample.heapSubmenu;

import javafx.scene.control.Button;
```

```
import javafx.scene.control.CheckBox;
import javafx.scene.control.TextField;
import javafx.scene.control.Label;
import jdk.internal.util.xml.impl.Input;
import sample.Main;
import javafx.application.*;
import javafx.fxml.*;
import javafx.scene.*;
import javafx.scene.layout.BorderPane;
import javafx.stage.*;

import javafx.fxml.*;

import java.util.InputMismatchException;


public class heapSubmenuController {

    private Main main;
    private CheckBox min;
    private CheckBox max;
    private Button create;
    private TextField input;
    private Label mismatch;

    @FXML
    private void digestInput() {
            String userInput = input.getText();
    }
    @FXML
    private void createSelected(){
        main.createHeap();
    }
    @FXML
    private void handleDoubleChecked(){
        if (min.isSelected()) {
            max.setSelected(false);
        }
        if(max.isSelected()) {
            min.setSelected(false);
        }
    }
}
```

Heap Solver: This subclass of Heap will act as the logic for the Heap class, and will take constraints from the submenu to accurately provide the logic for heap construction.

```
Package sample.HeapSubmenu
import javafx.scene.image.ImageView;
import sample.Main;
import javafx.fxml.*;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.TextField;
import javafx.scene.control.Label;
package sample.heapSubmenu;

import java.awt.*;
import java.util.*;
import javafx.*;
import javafx.animation.PathTransition;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.scene.shape.Polyline;

public class heapSolver {

    private int[] array;
    private int root;
    private int parent;
    private int leftChild;
    private int rightChild;
    private int level1 = 0;
    private int level2 = 1;
    private int level3 = 2;
    private int level4 = 3;
    private boolean heap;


    private int[] sanitizeInput(TextField userInput){
        String sanitize = userInput.getText();
        char[] charArray = sanitize.toCharArray();
        for(int x=0; x<sanitize;x++)
```

```
        {
            char c= charArray[i];
            array[i]= Character.getNumericValue(c);
        }
        return array;
    }
    private void createTreeCircles(int[] sanitizedInput) {
        Circle[] circles = new Circle[sanitizedInput.length];
        double shiftX= 100;
        double shiftY= 100;

        for (int x = 0; x < circles.length; i++) {
            circles[x] = new Circle(45, shiftX, shiftY); //root node is at top
            if(x%2 == 0)
            {
                shiftX some amount
            }
            else
            {
                shiftX some other amount for the right child
            }
    }
    private void createTreeLines(Circle[] c){
        Line[] lines = new Line[c.length*2];
        double startX= 100;
        double startY= 100;
        double stopX= 200;
        double stopY= 200;
        for (int x = 0; x < lines.length; i++) {
            lines[x] = new lines(startX,startY,stopX,stopY); //root node is at top
            if(x%2 == 0)
            {
                shiftX some amount
            }
            else
            {
                shiftX some other amount for the right child
            }




    }
}
```

```
private void createTreeLinesMovement(Circle[] circles){
   if(circles.length%2 ==0)
   {
      Polyline[] lines= new Polyline[circles.length*2]; //twice as many lines as circles if even
   }
   else
   {
      int linesTotal= circles.length*2;
      linesTotal=linesTotal-1;
      Polyline[] lines= new Polyline[linesTotal];
   }
}

private void moveLevel1(Circle[] c,Polyline[] p) {
   PathTransition transition1 = new PathTransition();
   transition1.setNode(circle[i]);
   transition1.setDuration(2);
   transition1.setPath(p[0]);
}
//repeat above method for the other shapes

private void createHeap(TextField userInput){
   userConvert();
   int nBy2= array.length/2;
   int k;
   int v;
   int j;

   for(int i=nBy2;i >=1; i--)
   {
      k=i;
      v= array[k];
      heap = false;

      while(( heap != true) && (2*k<=array.length))
      {
         j= 2*k;
         if(j<n)
         {
            if(array[j])<array[j+1])
            {
               j++;
            }
            if(v>=array[j])
```

```
                {
                    heap = true;
                }
                else {
                    array[k] = array[j];
                    k = j;
                }
            }

        }
        array[k]=v;
    }



  }

}
```

Horspool Submenu: This subclass of Horspool acts to create the input values of the Horspool class. The two values to be inserted are the main string and the substring to be found in the main string using the Horspool algorithm.

```
package sample.horspoolSubmenuController;

import sample.Main;

import javafx.fxml.*;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.TextField;
import javafx.scene.control.Label;
import jdk.internal.util.xml.impl.Input;
import sample.Main;
import javafx.application.*;
import javafx.fxml.*;
import javafx.scene.*;
```

```
import javafx.scene.layout.BorderPane;
import javafx.stage.*;

public class horspoolSubmenuController {

   private Main main;
   private TextField matchingField;
   private TextField substringField;

   private Button createHorspool;

   @FXML
   private void digestMatchingInput(){
      String matchMe = matchingField.getText();
   }
   @FXML
   private void digestSubstringInput(){
      String subString = substringField.getText();
   }
   @FXML
   private void createHor(){
      main.createHorspool();
   }

}
```

Horspool Solver: This subclass of Horspool will act as the logic for the Horspool class, and it will take constraints and values from the submenu class and perform substring matching based on the Horspool algorithm.

```
import java.util.*;

public class HorspoolSolver
{
   public char[] pattern //get input from text field in previous class
   public char[] abc //entire alphabet
   public int[] table=new int[abc.length];

   /**
```

```
    * This function prints the shift table
    */
   public void printTable()
   {
      for(int i=0;i<abc.length;i++)
      {
         System.out.print(abc[i]+ " ");
      }

      System.out.println("\n");
      for(int i=0;i<table.length;i++)
      {
         System.out.print(table[i]+" ");
      }
      System.out.println("\n");
   }
   /**
    * This function updates the change table based on the index with the appropriate shift
values
    */
   public int changeTable(char[] array,int j)
   {
      int result=0;
         if(array[j]=='A')
         {
            result=0;
         }
         //continue on for all letters
         return result;
   }
   /**
    * This is our shift table, using the algorithm in the book we fill and print the shift table with
the appropriate
    * Input: A character array pattern that is the pattern we are searching for.
    * Output: A integer array that is the shift table.
    */
   public int[] shiftTable(char[] pattern)
   {
      int m=pattern.length;
      for(int i=0;i<abc.length;i++)
      {
         table[i]=m;
      }
```

```
    printTable();
    for(int j=0;j<m-1;j++)
    {
       int result=changeTable(pattern,j);
       table[result]=m-1-j;
    }
    printTable();

    return table;
}
/**
 * Implements the string matching algorithm from the book.
 * Input: A character array that is the pattern, and a character array that is the user input.
 * Output: A value at the index if a match is found, if not, a -1 is returned.
 */
public int stringMatching(char[] pattern, char[] input)
{
    int[] table=shiftTable(pattern);
    int i=pattern.length-1;
    int n=input.length-1;
    int k;
    while(i<=n)
    {
       k=0;
       while(k<=pattern.length-1 && (pattern[pattern.length-1-k]==input[i-k]))
       {
          k=k+1;
       }
       if(k==pattern.length)
       {

          return i-pattern.length+1;
       }
       else
       {
          int result=changeTable(input,i);
          i=i+table[result];
       }

    }
    return -1;

}
/**
```

```
    * Changes a character array to all upper case characters
    * Input: A character array to shift to upper case
    */
   public static void toUpper(char[] array)
   {
      for(int i=0;i<array.length;i++)
      {
          array[i]=Character.toUpperCase(array[i]);
      }
   }
   public static void solveMe()
   {


      char[] pattern //get input from submenu

      boolean flag=true;



      String userInput=reader.nextLine();

      char[] split=userInput.toCharArray();

      toUpper(split);


      for(int i=0;i<split.length;i++)
      {
         if(split[i]==a || split[i]==c || split[i]==g || split[i]==t)
         {
               flag=true;

         }
      }

      Horspool h= new Horspool();
      int result=h.stringMatching(pattern,split);
   }
}
```

Prims Submenu: This subclass of Prims is the container for the constraints for Prims, and will present a simple menu of choices to show an example based on the selection chosen.

```
package sample.PrimsSubmenu;

import sample.Main;
import javafx.fxml.*;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.TextField;
import javafx.scene.control.Label;
import jdk.internal.util.xml.impl.Input;
import sample.Main;
import javafx.application.*;
import javafx.fxml.*;
import javafx.scene.*;
import javafx.scene.layout.BorderPane;
import javafx.stage.*;

public class primsSubmenuController {

   private Main main;
   private CheckBox example1;
   private CheckBox example2;
   private CheckBox example3;

   @FXML
   private void checkBox(){
      if (example1.isSelected()) {
         example2.setSelected(false);
         example3.setSelected((false);
      }
      if (example2.isSelected()) {
         example1.setSelected(false);
         example3.setSelected((false);
      }
      if (example3.isSelected()) {
         example1.setSelected(false);
         example2.setSelected((false);
      }
   }

   @FXML
```

```
    private void createP(){
        main.createPrims();
    }
}
```

Prims Solver: This subclass of Prims will act as the logic behind Prims, using the selected example from the submenu, it will show the correct path using Prim's algorithm.

```
import jdk.internal.util.xml.impl.Input;
import sample.Main;
import javafx.application.*;
import javafx.fxml.*;
import javafx.scene.*;
import javafx.scene.layout.BorderPane;
import javafx.stage.*;

public class primsSolver {

    private Main main;

    private ImageView image1;
    private ImageView image2;
    private ImageView image3;
    private boolean image1chosen;
    private boolean image2chosen;
    private boolean image3chosen;

    private void showImage(){
        if(image1chosen == true)
        {
            //show image 1
        }
        if(image2chosen == true)
        {
            //show image 2
        }
        if(image3chosen == true)
        {
            //show image3
        }
    }
```

```
}
```

Application Class: The application class is only used to ensure t hat the application is running.

```
package sample;/**

        import javafx.application.*;
        import javafx.fxml.*;
        import javafx.scene.*;
        import javafx.scene.layout.BorderPane;
        import javafx.stage.*;

public class Main extends Application {

    public static void showHeapSubmenu(){
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(Main.class.getResource("heapSubmenu/heapSubmenu.fxml"));
        BorderPane heapSubmenu = loader.load();
        mainLayout.setCenter(heapSubmenu);
    }
    public static void showHorspoolSubmenu(){
        FXMLLoader loader = new FXMLLoader();

loader.setLocation(Main.class.getResource("horspoolSubmenuController/horspoolSubmenu.fx
ml"));
        BorderPane horspoolSubmenu = loader.load();
        mainLayout.setCenter(horspoolSubmenu);
    }
    public static void showHuffmanSubmenu(){
        FXMLLoader loader = new FXMLLoader();

loader.setLocation(Main.class.getResource("huffmanSubmenuController/huffmanSubmenu.fx
ml"));
        BorderPane huffmanSubmenu = loader.load();
        mainLayout.setCenter(huffmanSubmenu);
    }
    public static void showPrimsSubmenu(){
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(Main.class.getResource("PrimsSubmenu/primsSubmenu"));
        BorderPane PrimsSubmenu = loader.load();
```

```
        mainLayout.setCenter(PrimsSubmenu);
    }
    public static void createHeap() {
    FXMLLoader loader = new FXMLLoader();
    loader.setLocation(Main.class.getResource("heapSubmenu/heapVisualize.fxml"));
    BorderPane heapVisualize = loader.load();
    mainLayout.setCenter(heapVisualize);
    }
    public static void createHuffman() {
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(Main.class.getResource("huffmanSubmenu/huffmanVisualize.fxml"));
        BorderPane huffmanVisualize = loader.load();
        mainLayout.setCenter(huffmanVisualize);

    }
    public static void createHorspool() {
        FXMLLoader loader = new FXMLLoader();

loader.setLocation(Main.class.getResource("horspoolSubmenuController/horspoolVisualize.fx
ml"));
        BorderPane horspoolVisualize = loader.load();
        mainLayout.setCenter(horspoolVisualize);
    }
    public static void createPrims(){
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(Main.class.getResource("PrimsSubmenu/primsVisualize.fxml"));
        BorderPane primsVisualize = loader.load();
        mainLayout.setCenter(primsVisualize);
}

    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root=
FXMLLoader.load(getClass().getResource("MainMenuNavigation/sample.fxml"));
        primaryStage.setTitle("Algorithm Visualizer");
        primaryStage.setScene(new Scene(root,1920,1080));
        primaryStage.show();

    }
    public static void main(String[] args)
    {
        launch(args);
    }
}
```

UI: This class will act as the graphical user interface for the user, and it will also act as the container for the buttons to navigate to appropriate screens. This class is the direct line of communication from user to system.

```
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.*;
import sample.Main;

import java.io.IOException;

public class mainMenuController {


    private Main main;
    @FXML
    private void heapButtonClicked() throws IOException {
        main.showHeapSubmenu();
    }
    @FXML
    private void horspoolButtonClicked() throws IOException {
        main.showHorspoolSubmenu();
    }
    @FXML private void huffmanlButtonClicked() throws IOException{
        main.showHuffmanSubmenu();
    }
    @FXML private void primsButtonClicked() throws IOException {
        main.showPrimsSubmenu();
    }
}
```

Note: Upon revision of our design face, these classes have been integrated into other classes, and have thus become obsolete. These changes have been documented.

Main Menu Button: This will serve as the return to home button for the application, and at any time the user may press this button to exit what they are doing and return back to the initial start screen.

Exit Button: This will serve as the quit application button, this button is available only on the main screen and will terminate the application..

Step Back Button: This will act as an undo button for the application and will step back one full step in the regarded algorithm.

Prims: This class acts as the overall container for our Prim's algorithm, and will be responsible for showing the pre-created examples based on constraints passed in the subclasses. Unlike the other classes and their respective algorithm demonstrations, these examples are pre-created to only demonstrate a basic foundation of movement in the algorithm.

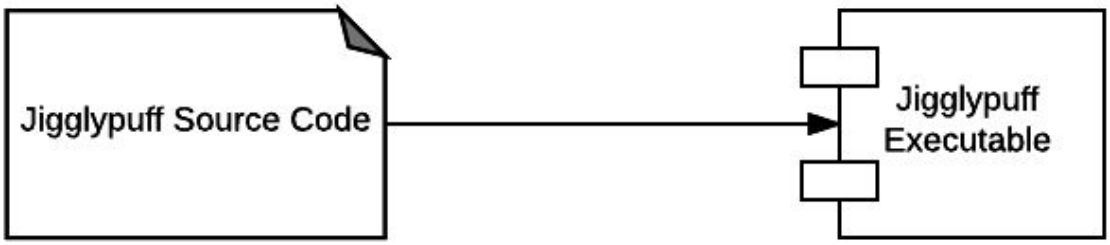
Horspool: This class acts as the overall container for the Horspool algorithm, and will be responsible for showing the visualization of Horspool's algorithm. This class will represent a qestring and a substring that will be searched within the main string. It will show the

visualization of the shift's of the substring and a shift table for each letter corresponding to the substring. It will use constraints passed in the subclasses.

Heap: This class acts as the overall container for the Heap Construction algorithm, and will be responsible for showing the visualization of the heap we construct. This class will represent a heap based on constraints created in the subclasses.

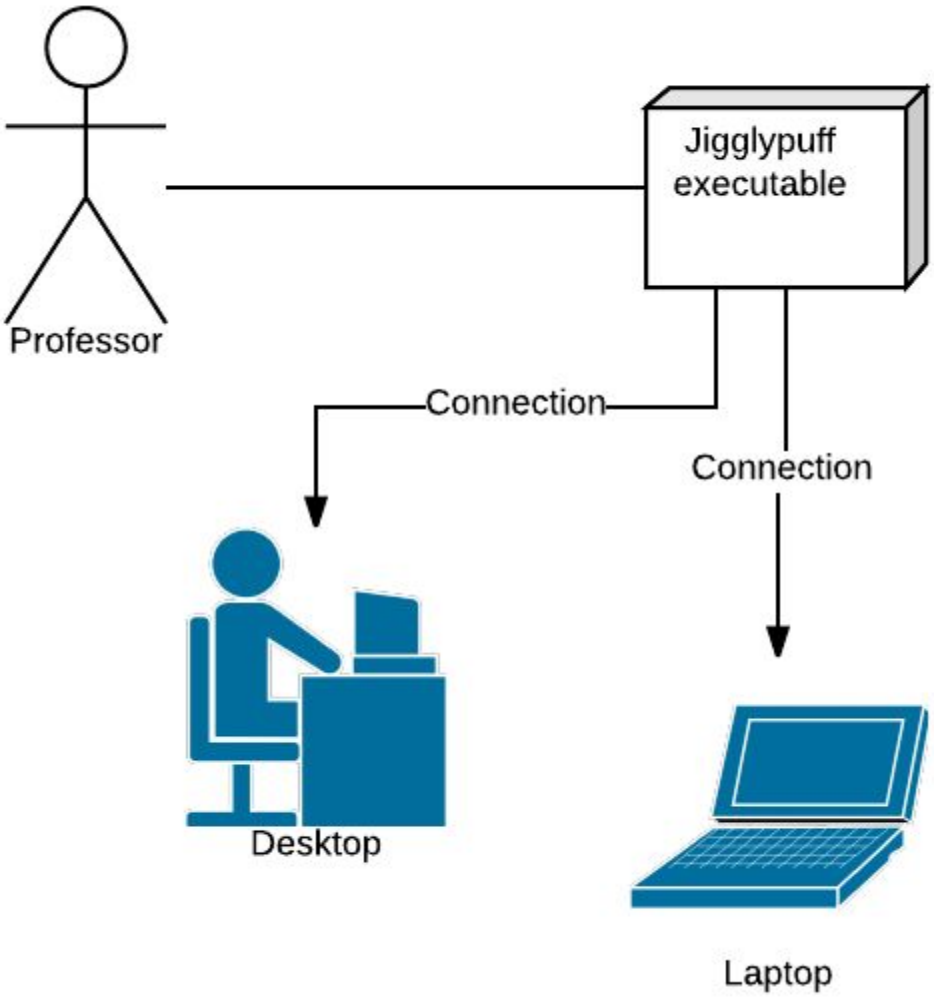
Huffman: This class acts as the overall container for the Huffman algorithm, and will be responsible for showing the visualization of Huffman's algorithm. This class will represent a number based in a tree in it's binary representation based on the Huffman's algorithm.

Component Diagram



The component diagram only has one component connected to the source code because it is a standalone application for the professor to use and is not meant for any client/server communication or maintenance. All source code, which includes all classes shown in the class diagram, will be implemented in the executable.

Deployment Diagram



Our deployment diagram consists of the professor using either their laptop or desktop to run the executable in class as an aid to help the students have a better understanding of the traversal of the separate algorithms.

Appended Material

Metrics

For our metrics, we decided to keep track of two metrics, Volatility and Size.

Volatility: We kept a record on how frequently all of the documents had a change implemented by one of our the team members. To the right is how often the requirements changed. Totalling 141 changes in 6 days of work, Averaging 23.5 changes each work day.

Date	# Changes
01/31/17	11
02/01/17	2
02/03/17	2
02/05/17	90
02/06/17	35
02/07/17	1
Total	141

Size (KLOC): We will be keeping track of the LOC the program contains during each of the steps of the implementation workflow. With the data gathered we will be computing the Errors per KLOC and Defects per KLOC. We have the assumptions that the project will take 2 KLOC with at least 50 errors per 1 KLOC.